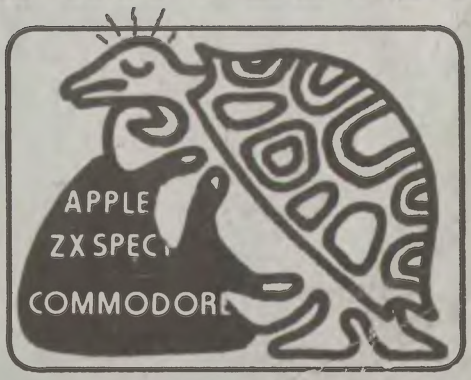
A

Logo

PROGRAMOZÁSI NYELV



A Logo programozási nyelv

A Logo programozási nyelv

Műszaki Könyvkiadó, Budapest, 1986

Az első rész

Turcsányiné Szabó Márta munkája

A második rész

Dietrich Senftleben: Programmieren mit Logo c. könyve alapján készült.

Fordította:
Bendl János

© Vogel-Buchverlag, 1983

A függelék összeállításában közreműködött:

Csanádi György
Turcsányiné Szabó Márta

Lektorálta:

Pogány Gábor

Hungarian translation

© Műszaki Könyvkiadó, Budapest, 1986.

ETO: 519.682 LOGO
800.92 LOGO

ISBN: 963 10 6841 2

Felelős szerkesztő: **Votisky Zsuzsa**

Tartalom

Előszó a magyar kiadáshoz 11

Bevezetés 12

I. RÉSZ

1. Teknőc-grafika 15

1.1. A Teknőc mint geometriai kifejezőeszköz 16
1.2. Milyen nyelven értenek a Teknőcök? 18
1.2.1. A Teknőc-utasításpárok 19
1.3. Rajzolás a Teknőccel 21
1.3.1. Hogyan fogjunk hozzá egy elképzelt ábra megrajzolásához 32
1.4. Grafikai kellékek 46
1.4.1. Animáció 46
1.4.2. Véletlenszámok alkalmazása 51
1.4.3. Színezés 52
1.5. A Teknőc-grafika hatékonysága 54
1.5.1. Ismétlések 54
1.5.2. Síkkitöltő ábrák 58
1.5.3. Csillagok és spirálok 64
1.5.4. Rekurzív görbék 72

II. RÉSZ

2. Üdvözöljük a Logoban 79

2.1. Közvetlen utasítások 79
2.1.1. A számítógép mint zsebszámológép 79
2.1.2. Amit a zsebszámológép már nem tud 81
2.1.3. Ismerkedés a műveletekkel – Szavak és listák feldolgozása 83

2.1.4. Hibajavítás . . . 85
2.1.5. A TYPE és a PRINT utasítás . . . 86
2.1.6. Összefoglalás és gyakorlatok . . . 86
2.2. Egyszerű nyomtató programok . . . 88
2.2.1. Egy versike . . . 88
2.2.2. Címzés . . . 89
2.2.3. Geometriai ábrák és nyomtatott képek . . . 90
2.2.4. Minden megismételhető . . . 90
2.2.5. Programok módosítása a Logo editorral . . . 91
2.2.6. Összefoglalás és gyakorlatok . . . 92
2.3. Bővülő programok . . . 93
2.3.1. Programok programokat hívnak . . . 93
2.3.2. Programok bemenő adatokkal . . . 94
2.3.3. Összefoglalás és gyakorlatok . . . 97
2.4. Grafikai kirándulás . . . 98
2.4.1. Séta a képernyőn . . . 98
2.4.2. Nagyítás és kicsinyítés . . . 101
2.4.3. Csillagköd . . . 106
2.4.4. Összefoglalás és gyakorlatok . . . 107
2.5. Nyomtatót használunk, a programokat mágneslemezen tároljuk . . . 109
2.6. Műveletek definiálása . . . 110
2.6.1. Kimenet: OUTPUT . . . 110
2.6.2. Választás: ITEM . . . 115
2.6.3. Véletlen: RANDOM . . . 115
2.6.4. Véletlenregény . . . 117
2.6.5. Összefoglalás és gyakorlatok . . . 119
2.7. Igaz és hamis vizsgáló szavak . . . 120
2.7.1. Vizsgáló szavak a szókészletben . . . 120
2.7.2. Felhasználó által definiált vizsgáló szavak . . . 122
2.7.3. Összefoglalás és gyakorlatok . . . 124
2.8. Két fontos vezérlő szó . . . 125
2.8.1. A programelágazás . . . 125
2.8.2. A STOP utasítás . . . 129
2.8.3. A programok önmagukat hívják ellenőrzés mellett . . . 130
2.8.4. Összefoglalás és gyakorlatok . . . 135
2.9. Rekurzív műveletek . . . 136
2.9.1. Önmagukat hívó műveletek . . . 136
2.9.2. Összefoglalás és gyakorlatok . . . 139
2.10. Ugrás és végrehajtás . . . 141
2.10.1. Ugrási utasítás . . . 141
2.10.2. Értékadás címmel és tartalommal . . . 142
2.10.3. Programciklus . . . 145
2.10.4 Összefoglalás és gyakorlatok . . . 146

3. Logo haladóknak 148

3.1. Lokális és globális változók 148
3.2. A munkatár kezelése 151
3.2.1. Csoportos törlés, írás és elrejtés 151
3.2.2. Ha kicsi a munkatár 155
3.3. Képernyő- és kurzorvezérlés 157
3.4. Az adatbevitel módjai 159
3.5. Vezérlő szerkezetek 163
3.5.1. A Logoban alkalmazható vezérlő szerkezetek 163
3.5.2. Más programnyelvek vezérlő szerekezetei a Logoban 167
3.5.3. Hibakimenetek definiálása 177
3.6. Strukturált listák 180
3.7. A programok programokkal dolgoznak 187

4. Példák nagyobb programokra 190

4.1. Magyar nyelvű Logo 190
4.2. Adatfeldolgozás 195
4.3. Függvények rajzolása 203
4.4. Az indexes (tömb-) változók használata 215
4.4.1. Felhasználó által definiált táblázatfunkció 216
4.4.2. Mátrixműveletek 220
4.4.3. Tömbrendezések Bubblesort 223

III. RÉSZ

FÜGGELÉK 227

1. Az LCSI Apple Logo alapfunkciói 229

1.1. Aritmetikai műveletek 229
1.2. Képernyőfunkciók 230
1.3. Definiálás és editálás 231
1.4. Tulajdonságlisták 232
1.5. Ki- és bemeneti utasítások 232
1.6. Listákat és szavakat kezelő funkciók 233
1.7. Vezérlő szerkezetek 234
1.8. Lemezről olvasás és lemezre írás 236
1.9. Logikai műveletek 237
1.10. Vizsgáló műveletek 237
1.11. Rendszerfunkciók 238
1.12. Teknőc-grafika 238
1.13. A munkaterület kezelése 241
1.14. Értékadás (változók definiálása) 243
1.15. További fenntartott Logo szavak 243

2. A Commodore 64 Terrapin Logo leírása 245

2.1. Bevezetés 245
2.1.1. Funkciók szerkesztőállapotban 246
2.1.2. Funkciók parancsállapotban 247
2.1.3. Funkciók futóállapotban 248
2.2. Képernyőkezelés 250
2.2.1. Információk 250
2.2.2. Rajzmező – Teknőc-utasítások 251
2.2.3. Rajzmezőt kezelő utasítások 253
2.2.4. Szövegmezőt kezelő utasítások 254
2.2.5. Írás a szövegmezőre 255
2.3. Műveletvégzés 257
2.3.1. Műveleti jelek 257
2.3.2. Aritmetikai függvények 258
2.3.3. Véletlenszámok 258
2.3.4. Bites műveletek 259
2.4. Szövegkezelés 259
2.4.1. Szövegkezelő utasítások 260
2.4.2. ASCII konverziók 261
2.5. Vezérlésirányítás 261
2.5.1. Feltételes utasítások 261
2.5.2. Logikai utasítások 262
2.5.3. Feltétel nélküli vezérlési utasítások 263
2.5.4. A hibakeresés eszközei 264
2.6. Bemenő és kimenő jelek 264
2.6.1. Bemenő jeleket kezelő utasítások 264
2.6.2. Bemeneti egységek jeleit kezelő utasítások 265
2.6.3. Kimeneti egységek jeleit kezelő utasítások 266
2.7. Definiálás, javítás és törlés 267
2.7.1. Eljárásokra vonatkozó utasítások 267
2.7.2. Változókra vonatkozó utasítások 269
2.8. Lemezkezelés 270
2.9. Sprite-kezelés 271
2.10. Speciális utasítások 272
2.11. Hibaüzenetek 274

3. A Snail Logo Sinclair ZX Spectrumra 278

3.1. A Snail Logo használata 278
3.1.1. Bevezetés 278
3.1.2. Példák 279
3.1.3. Első lépések 280
3.1.4. Következő lépések 281
3.1.5. Változók 283

3.1.6. Eljárások és paraméterek 285
3.1.7. Rekurzív eljárások 287
3.1.8. Számítások 289
3.2. A Snail Logo operációs rendszere 289
3.2.1. Bevezetés 289
3.2.2. A Snail Logo rendszer betöltése 290
3.2.3. Az operációs rendszer lehetőségei 290
3.3. Összefoglalás 293
3.3.1. A Snail Logo utasítások összefoglalása 293
3.3.2. Megjegyzések az utasítások használatához 296
3.3.3. Működési korlátok 297
3.3.4. Hibaüzenetek 297

4. Az egyes fejezetek szimbólumai és a hozzá tartozó programok 300

5. Fordítóprogram 304

6. Személyi számítógépek Logo változatainak táblázata 312

Irodalom 317

Előszó a magyar kiadáshoz

E kötetben a lehetőségekhez mérten igyekeztünk széles felhasználói körnek hozzáférhetővé tenni a Logo programozási nyelvet. Könyvünk jórészt D. Senftleben azonos című műve alapján készült.

A Logo egyik legnevezetesebb tulajdonsága a Teknőc-grafika, ezért ennek a témakörnek egy külön, bevezető fejezetet szenteltünk.

D. Senftleben programpéldái Apple IIe számítógépre készültek. Tekintettel azonban arra, hogy hazánkban a ZX Spectrum és a Commodore 64 jóval elterjedtebb gépek, külön függelékben közöljük a ZX Spectrum Snail Logo, ill. a C-64 Terrapin Logo leírását.

A függelékben egy Logo fordító programot is közlünk, amely lehetővé teszi, hogy a könyv Apple IIe-re szóló példáit C-64-en is ki lehessen próbálni, ill. futtatni. Továbbá egy összehasonlító táblázatot is mellékelünk a különböző Logo változatok kulcsszavairól.

Az átdolgozásban való segítségért a magyar szerzőn és a lektoron kívül Mag Pálnak mond köszönetet

A Kiadó

Bevezetés

A „Logo" az ógörög „Logos" szóra utal, ami magyarul értelmet, tudományt és persze logikát jelent. A Logo filozófiája, vagyis alapgondolata a modulszerűen felépített programozás, amelyben az egyes elemeket a programozó készíti el. A programírás, ill. -építés így logikai egységekre bontható; tisztán alkalmazható a bottom-up és top-down (alulról fölfelé, ill. fölülről lefelé) való részfeladatra bontás elve.

A Logo programozási nyelvet az Egyesült Államokban a Massachusetts Institute of Technology intézetében Seymour Papert irányításával fejlesztették ki. Az volt a cél, hogy minél hatékonyabb kommunikációs lehetőséget biztosítsanak az ember és gép között, hogy minél hatékonyabban ismertessék meg a kezdőkkel a számítógép megkövetelte gondolkodásmódot.

Éppen ezért a Logo kiválóan használható az oktatásban, matematikai, természettudományos tanulmányokban. Egyébként nem tekinthető csupán gyermek programozási nyelvnek, mert az oktatás bármely szintjén sokféle távlatot nyit.

A Logo programozási nyelvben hozták létre elsőként a Teknőc-grafikát, ami azóta egy új tudomány, A Teknőc-geometria megszületéséhez vezetett.

A modul rendszerű építkezés lehetővé teszi az anyanyelven, tehát a magyarul való kommunikációt is. A Logo nyelvjárásai nem sokban különböznek egymástól. Inkább csak arról van szó, hogy egyes gépeken bővebb, másokon pedig szűkebb az alap-utasításkészlet, vagy mások a rövidítések. A BASIC nyelvtől való gyökeres különbözőséget már a tanulás viszonylag kezdeti fokán könnyű észrevenni. Tehát BASIC programokat nem érdemes átfordítani, inkább a problémából kiindulva kell a Logo-beli megoldást keresni.

I. RÉSZ

1. Teknőc-grafika

1. Teknőc-grafika

A teknősök a csuszómászók rendjébe tartoznak. Valamennyi teknősféle négylábú, és legfőbb jellemvonása, hogy széles és rövid törzsét páncél övezi úgy, hogy csak a fej, láb és farok látszik ki alóla. A teknősök lomha, lassú állatok. Értelmi képességeik igen alacsony szintűek.

(A Révai lexikon nyomán)

Mi köze a teknősbékának a számítógéphez? Alapjában véve semmi. Csupán egy találó elnevezés, ami a számítógéppel irányítható kis rajzoló robot és a Teknőc formai hasonlóságán alapszik.

A robot-Teknőc a számítógép klaviatúráján keresztül utasítható a számára „érthető" feladatok elvégzésére. Adott távolsággal előre, hátra tud menni; adott fokkal jobbra, balra tud fordulni; tollat (ami a hasara van erősítve)

felemeli, leereszti, más színűre cseréli; némelyik még fényérzékelőkkel is fel van szerelve vagy éppenséggel világít, sőt beszél. Mindenesetre közös vonásuk, hogy egy jókora rajzlapon mozogva csodálatos rajzokra képesek.

A Teknőc modellje

A robot-Teknőc szimulációja a képernyő-Teknőc. △

A képernyőn látható rajzmező síkja a rajzlap, a közepén látható háromszögletű alakzat a Teknőc. Vastagabb oldalát a hátának, a hegyesszöget pedig az orrának tekintjük. Amerre elhalad, nyomot hagy. A vonal színe a számítógép és a tv (vagy monitor) lehetőségeitől függ.

1.1. A Teknőc mint geometriai kifejezőeszköz

„A geometria pontok, vonalak, felületek és testek alakjával, nagyságával, egymáshoz való viszonyával és mérésével foglalkozik." – olvashatjuk Obádovics Gyula matematikai összefoglalójában.

Az egyes geometriai rendszerek az alapfogalmakról szóló tételekre és meghatározásokra épülnek fel. Ugyanakkor egy geometriai rendszeren belül, az idomok meghatározása többféleképpen történhet. Most csak a síkra szorítkozunk.

Az *euklidészi* geometria a sík elemeinek a pontot, egyenest és a szöget tekinti, így a síkbeli alakzatokat ezek kapcsolatai fejezik ki. A pont ebben a megközelítésben statikus. Euklidész az axiomatikus logikai stílust képviselte – mint tudjuk – először bármilyen tudományban.

Descartes a sík elemének a pontot tekintette mégpedig úgy, hogy helyzetét a sík két, irányított, egymásra merőleges, egységgel ellátott egyeneséhez (a koordinátatengelyhez) viszonyította. Így a sík bármely pontjának helyzetét a koordináta-rendszer két tengelyétől mért előjeles távolsággal határozta meg. A síkbeli alakzatokat az alakzat pontjainak a koordináta-rendszerhez viszonyított egyenleteivel adta meg. A descartes-i geometria analitikus, algebrai stílusú.

Ismerünk más koordináta-rendszert is, ilyen pl. a *polárkoordináta-rendszer.* Ebben egy síkbeli pont helyzetének meghatározásához egy 0 pontot (pólus), és egy 0-ból kiinduló félegyenest (polártengely) kell felvenni. A sík egy pontjának polárkoordinátáját a pont pólustól mért távolsága és a pólustól a ponthoz futó egyenes polártengellyel bezárt szöge határozza meg. A síkbeli alakzatokat a pontjainak polárkoordinátákkal kifejezett egyenletei adják meg.

A *Teknőc* geometriailag nem más, mint egy állapottal rendelkező pont a síkon.

Az állapotot meghatározza a pozíció: a rajzmező síkján elfoglalt hely (Descartes-koordináta) és az irány: északhoz (felfele) képest az óramutató járásával megegyezően pozitívan mért elfordulási szög (polárszög).

A síkbeli alakzatokat a Teknőc állapotváltoztató utasításaival adhatjuk meg. Az alakzat a Teknőc által megrajzolt ábra.

Állapotváltoztató utasítások:

FORWARD hossz: hatására a Teknőc az adott hosszúsággal előre megy.

BACK hossz: hatására a Teknőc az adott hosszúsággal hátra megy.

(Ezek az utasítások a Teknőc *pozícióját* változtatják meg, míg az irányát változatlanul hagyják.)

RIGHT fok: hatására a Teknőc az adott fokkal jobbra fordul.

LEFT fok: hatására a Teknőc az adott fokkal balra fordul.

(Ezek az utasítások csupán a Teknőc *irányát* változtatják meg, pozícióját változatlanul hagyják.)

A Teknőc a konstruktív, vagyis a „létrehozó" stílust képviseli. Például a négyzet definíciója a Teknőc-geometriában:

```
TO NEGYZET1 :HOSSZ
 FORWARD :HOSSZ RIGHT 90
 FORWARD :HOSSZ RIGHT 90
 FORWARD :HOSSZ RIGHT 90
 FORWARD :HOSSZ RIGHT 90
END
```

A síkbeli alakzatok definíciója egyben létrehozásuk algoritmusa.

Ugyanakkor egy Teknőc-program bizonyos fokig egy differenciál egyenletnek feleltethető meg.

Például RIGTH 1 FORWARD 1 csak azt a különbséget adja meg, hol van *most* a Teknőc és hol lesz *egy pillanat múlva.*

Hasonlítsuk össze a kör következő meghatározásait!

Az *euklidészi geometriában* a kör (körvonal) a sík mindazon pontjainak mértani helye, amelyeknek távolsága egy adott ponttól állandó.

A *descartes-i geometriában* $(x-a)^2+(y-b)^2 = R^2$.

A *Teknőc-geometriában* fordulj kicsit előre kicsit ...

A *kicsit* nevű állandó mértéke adhatja meg a különböző nagyságú köröket. Konkrét definíciót később adunk. (Ez egyébként egy sokszög leírása volt, de elég kicsi lépések esetén a képernyő szempontjából körnek tűnik.)

Mivel a Teknőc-tevékenységek megegyeznek az emberi mozgásmechanizmus egyes elemeivel, így a definíciókat könnyűszerrel magunk is realizálhatjuk, ill. megérthetjük a Teknőc megszemélyesítésével. E miatt lehetséges, hogy a Teknőc-geometria igen korán, gyermekeknek is tanítható.

Összefoglalva, az állapottal rendelkező pont és az állapotváltoztató utasítások a síkon (és akár a téren) együttesen jelentik a *Teknőc-geometria elemeit.*

1.2. Milyen nyelven értenek a Teknőcök?

A butuska teknősbékákat talán lehet idomítani, és nyilván mindegy, hogy az idomító milyen nyelven érteti meg magát velük. A képernyő- és robot-Teknőcök azonban csak speciális vezérlőjelekkel irányíthatók. Egyelőre a mesterséges intelligencia tudománya még nem tart ott, hogy természetes nyelven vezérelhető robotok készüljenek. Ezért, a számítógép klaviatúráján keresztül beírható programozási nyelvek maradnak, de ezek közül csak azokat érdemes figyelembe venni, amelyek ezt a kommunikáció lehető legmagasabb szintjén művelik, mert így ezek a lehető legtermészetesebb kifejezési eszközök.

Elsőként a Logo programozási nyelvbe építették be a Teknőc-vezérlő utasításokat. A Logo nyelvet a 60-as évek vége óta fejlesztették ki Seymour Papert irányításával az Amerikai Egyesült Államokban, a gyermekek egyszerű számítógépes kommunikációjának eszközéül.

Könyvünk 4 gépi nyelvjárásának megfelelően

- Apple:
- ZX Spectrum, Snail Logo:
- ZX Spectrum, Spectrum Logo:
- Commodore 64, Terrapin Logo:

szimbólumokat alkalmazzuk.

Egyes gépeken több nyelvjárás is elterjedt, mint a ZX Spectrumon, vagy az Apple gépen, amelyre létezik többféle, pl. a Terrapin gyártmányú Logo is, amely nagyon hasonlít a Commodore 64-en levő, szintén Terrapin cégérű változatra. Mindegyik Logo nyelvjárás valamennyire eltér a másiktól, de egyben megegyeznek: mindegyik rendelkezik Teknőc-grafikával.

Érdemes megemlíteni azokat a nyelveket is, amelyeknek egyes változatait hasonló Teknőc-utasításokkal bővítették ki:

- SUPER BASIC;
- COMAL;
- FORTH;
- PILOT;
- SMALLTALK;
- UCSD PASCAL.

Akik nem tudnak hozzáférni a Logo egyik nyelvjárásához sem, vagy a felsorolt nyelvek valamelyike közelebb áll hozzájuk mint kifejezési eszköz, azok is ugyanúgy megismerkedhetnek a fejezet programjaival, gondolataival, hiszen ebben a fejezetben kimondottan a Teknőc-grafikával foglalkozunk.

1.2.1. A Teknőc-utasításpárok

Párokba állítjuk az ellentétes, ill. kérdező–utasító parancsokat, hogy az utasítások közötti eligazodást segítsük.

FORWARD, FD ↔ BACK, BK

FORWARD ↔ BACK.

RIGHT, RT ↔ LEFT, LT

RIGHT ↔ LEFT.

Az ismert utasításpárokon kívül a legtöbb Teknőc a következőket is értelmezi:

PENDOWN ↔ PENUP

DOWN ↔ UP.

Ennek hatására a Teknőc a rajztollat leereszti, ill. felemeli és így nyom nélkül is tud haladni.

HIDETURTLE, HT ↔ SHOWTURTLE, ST

SNAIL ↔ NSNAIL.

Ezzel a Teknőc láthatatlanná tehető, de ugyanúgy képes tevékenykedni, mint egyébként.

SETHEADING, SETH ↔ HEADING.

Ezzel beállítható, ill. lekérdezhető a Teknőc iránya (a pozíció nem). Paramétereként alkalmazható még pl. a TOWARDS utasítás, amely megadja, milyen iránnyal kell a Teknőcnek rendelkeznie, hogy az adott xy koordináták felé nézzen.

SETX, SETY, SETPOS ↔ XCOR, YCOR, POS, POSITION

SETX, SETY, SETPOS ↔ XCOR, YCOR, POS

SETX, SETY, SETXY ↔ XCOR, YCOR

POSITION.

Ennek segítségével beállítható, ill. lekérdezhető a Teknőc pozíciója (az irány nem). Az origó a rajzmező közepe.

További hasznos grafikai utasítások:

HOME	CENTRE	HOME	A Teknőc a rajzmező közepére megy, iránya észak lesz.
CLEAN		CLEAR SCREEN	Letörli a rajzmezőt (a Teknőc marad a helyén).
CLEARSCREEN		DRAW	Az előbbi kettő együttes hatása.

A továbbiakban a példaprogramok utasításai a Commodore 64 Terrapin Logo utasításkészletének megfelelően szerepelnek. A programok könnyűszerrel átírhatók más változatra is az előbbiek alapján. Ha akad olyan utasítás a listában, amelyről itt nem szóltunk, akkor a függelék táblázataiban nézzen utána az Olvasó.

1.3. Rajzolás a Teknőccel

Először kísérletezzünk a FORWARD, BACK, RIGHT, LEFT utasítások alkalmazásával, s egyben a Teknőc-lépések képernyőhöz viszonyított nagyságát próbálgassuk.

FORWARD 50

A helyköz nagyon fontos, mert a Logo ezzel választja szét az értelmezésre váró szavakat (utasításokat) és számokat (paramétereket). Ha begépeltünk egy utasítást, még nem történik semmi. Az utasítás hatását csak a <RETURN>

gomb lenyomásával válthatjuk ki.

Ha üres rajzmezővel szeretnénk tovább dolgozni, inkább a DRAW parancscsal töröljünk, mert ez egyben kezdőállapotba helyezi a Teknőcöt.

- Határozzuk meg pontosan a rajzmező Teknőc-hosszal kifejezhető méretét! A rajzmező közepétől mérve egyenlő távolság-e fel, le, ill. jobbra, balra a rajzmező széle?

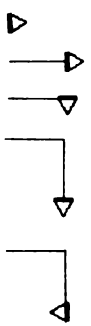
Útmutatás: Segítségünkre lehet ennek megállapításában a NOWRAP utasí-

tás, amelynek hatására a Teknőctől a rajzmező széléig mért távolság-

nál nagyobb paraméterű FORWARD, ill. BACK utasítás hibát jelez, és nem hajtódik végre.

- Próbáljunk rajzolni a Teknőccel!

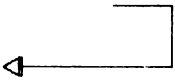
grafika program



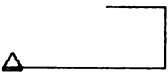
```
RIGHT 90
FORWARD 20
RIGHT 90
FORWARD 20
RIGHT 90
```

grafika

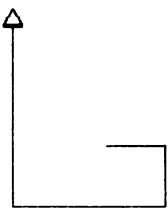
program



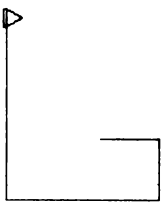
```
FORWARD 50
```



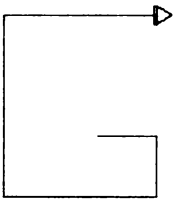
```
RIGHT 90
```



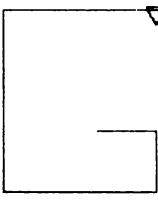
```
FORWARD 60
```



```
RIGHT 90
```

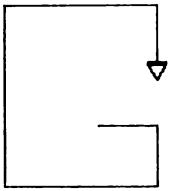


```
FORWARD 50
```

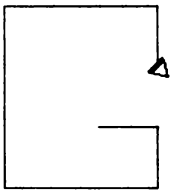


```
RIGHT 90
```

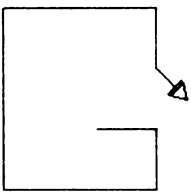
grafika program



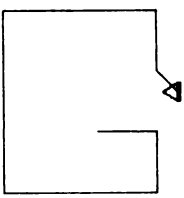
```
FORWARD 20
```



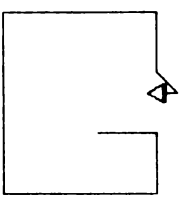
```
LEFT 45
```



```
FORWARD 10
```

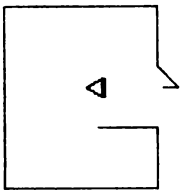


```
RIGHT 135
```

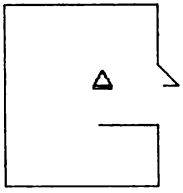


```
FORWARD 5
```

grafika program

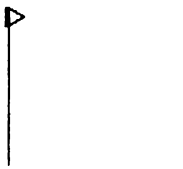


```
PENUP
FORWARD 20
```

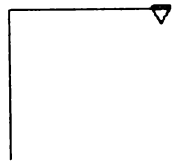


```
RIGHT 90
```

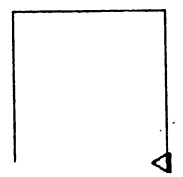
- Rajzoljunk négyzetet!



```
FORWARD 50 RIGHT 90
```

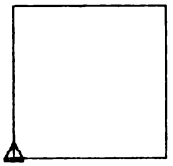


```
FORWARD 50 RIGHT 90
```



```
FORWARD 50 RIGHT 90
```

grafika program



```
FORWARD 50 RIGHT 90
```

Az ismétlések legegyszerűbb megfogalmazási módja a REPEAT utasítás alkalmazásával történhet:

```
REPEAT 4 [FORWARD 50 RIGHT 90]
```

(Sose felejtsünk helyközt tenni minden különálló szó szám, jel közé!)

Ezzel kifejeztük a négyzet definícióját. Most tanítsuk meg a Teknőcöt a négyzetrajzolásra.

```
TO NEGYZET                          } az eljárás feje
  REPEAT 4 [FORWARD 50 RIGHT 90]    } az eljárás törzse
                                      (több sorból állhat)
END                                 } az eljárás vége
```

Az eljárás fejének beírása után a <RETURN> gombot lenyomva szerkesztő állapotba jutunk. Az eljárás törzsének minden sora után nyomjuk le a <RETURN> gombot. Nem minden Logo nyelvjárás követeli meg az END beírását (C= esetében nem kell). A <CTRL> C vagy <RUN/STOP> billentyűk lenyomásával az eljárás definiálttá válik.

Most már a Teknőc érti a NEGYZET utasítást is. A NEGYZET begépelése és a <RETURN> gomb lenyomása után a Teknőc a pillanatnyi állapotából kiindulva egy 50 oldalhosszúságú négyzetet rajzol. Bárhol is van a rajzmezőn, bármilyen irányba is néz, mindig 50 oldalhosszúságú négyzetet rajzol.

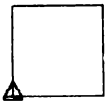
Ha a NEGYZET eljárás fejrészében egy bemeneti paramétert definiálunk, akkor az adott paraméternévvel már különböző oldalhosszúságú négyzetet is rajzolhatunk.

EDIT NEGYZET utasítás segítségével jutunk vissza a szerkesztő üzemmódba, és javíthatunk, módosíthatunk. A javítás helyét a (CRSR ⇑, ⇓, ⇒, ⇐) gombok lenyomásával érhetjük el. Törlésnél a kurzort mindig a törlendők utáni pozícióra állítjuk és a <DEL> gomb lenyomásával töröljük a nemkívánt jeleket. A lenyomott gombokon levő betűk mindig a kurzor (villogó négyzet) helye elé kerülnek beszúrásra.

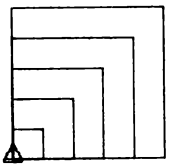
A paraméteres NEGYZET eljárás végső alakja a következő:

```
TO NEGYZET :HOSSZ
 REPEAT 4 [FORWARD :HOSSZ RIGHT 90]
END
```

A paraméter előtt levő kettőspont azt jelöli, hogy a HOSSZ nevű változó *tartalmáról* van szó tehát a tartalom lesz a paraméter. Az eljárás fejében definiált paraméterek csak az eljárás működése közben léteznek. A NEGYZET 30 egy 30 oldalhosszúságú négyzetet rajzol.



- Próbáljuk megrajzolni a következő ábrát:

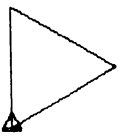


- Készítsünk szabályos háromszöget rajzoló eljárást! Ne feledjük, külső szögeket mérünk!

Megoldás:

```
TO HAROMSZOG :HOSSZ
 REPEAT 3 [FORWARD :HOSSZ RIGHT 120]
END
```

Például: HAROMSZOG 40 utasítás hatása

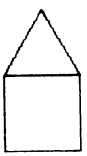


- Írjunk eljárást változó oldalhosszúságú szabályos sokszög létrehozására!

Útmutatás: Legyen az eljárás paramétere az oldalhosszúság és a csúcsok száma.

- Építsünk a meglevő elemekből tetszetős házat, várat ... stb.

Például:



Ez a ház 2 részből, a ház törzséből (négyzet) és a ház tetejéből (háromszög) áll. Így felhasználhatjuk a NEGYZET és a HAROMSZOG eljárásokat a megrajzolásához.

Hogyan illesszük a tetőt a törzsre?

Ime a ház törzse.

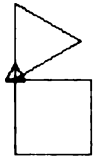
```
NEGYZET 25
```

Ide kell a tetőt tenni!

```
FORWARD 25
```

De nem így!

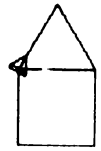
```
HAROMSZOG 25
```



No most már jöhet a tető. — 2

```
RIGHT 30
```
— 1

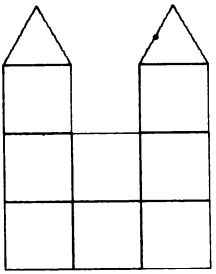
```
HAROMSZOG 25
```

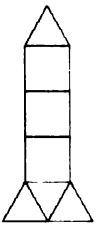


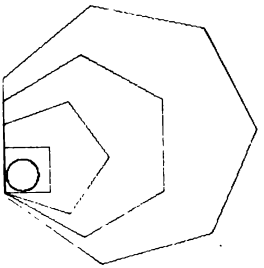
És íme egy tetszőleges nagyságú ház megrajzolásához szükséges eljárás:

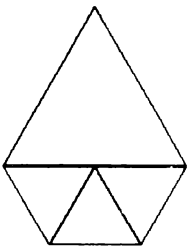
```
TO HAZ :MERET
 NEGYZET :MERET
 FORWARD :MERET
 RIGHT 30
 HAROMSZOG :MERET
 LEFT 30
 BACK :MERET
END
```

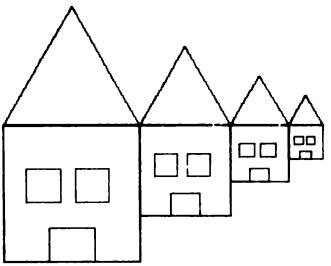
Néhány mutatós ábra:









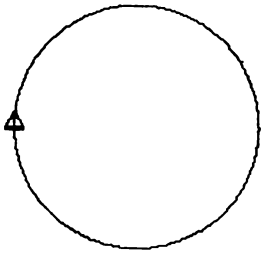


Nyilván nemcsak egyenes szakaszokból álló ábrákat tudunk a Teknőccel rajzolni, hanem görbét is.

A görbe vonalak apró egyenes szakaszokból (pontokból) rakhatók össze úgy, hogy a képernyő szempontjából görbének tűnjenek.

Például a

REPEAT 360 [FORWARD 1 RIGHT 1]



utasítássorozat kört rajzol, amelynél a Teknőc pontosan 360 lépést tesz meg és 360 fokot fordul!

- Írjunk olyan eljárást, amelynél a kör méretét lehet befolyásolni!

Például:

```
TO KOR1 :HOSSZ :SZOG
 REPEAT 360 / :SZOG [FORWARD :HOSSZ RIGHT :SZOG]
END
```

- Érdemes tovább változtatgatni az eljárás paraméterértékeit. Először az egyik paraméterértéket fixen hagyva a másikat változtassuk, majd fordítva.
- Írjunk olyan körrajzoló eljárást, amelynél a Teknőc pillanatnyi pozíciója a kör középpontja, és az eljárás paramétere a kör sugara.

Ehhez egy kicsit számolni kell:

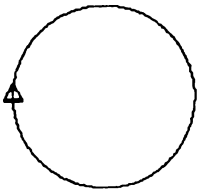
$2 * \pi *$ sugár = a kör kerülete = $360 *$ HOSSZ

HOSSZ = sugár $* \pi/180 \cong$ sugár $* 0{,}0175$.

Így az eljárás a következő:

```
TO KOR2 :SUGAR
 REPEAT 360 [FORWARD :SUGAR * 0.0174 RIGHT 1]
END
```

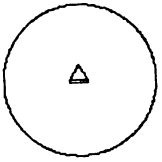
KOR2 30 hatása



Innen már csak egy lépés a végső megoldás:

```
TO KOR :SUGAR
 PENUP
 FORWARD :SUGAR
 RIGHT 90
 PENDOWN
 KOR2 :SUGAR
 LEFT 90
 PENUP
 BACK :SUGAR
 PENDOWN
END
```

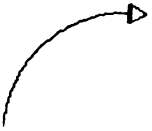
KOR 20 hatása



- Írjunk körívrajzoló eljárást!

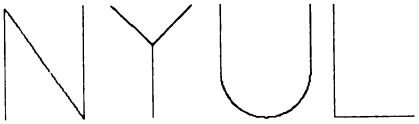
```
TO JOBB.IV :FOK :SUGAR
 REPEAT :FOK [FORWARD :SUGAR * 0.0174 RIGHT 1]
END
```

JOBB.IV 90 50 hatása



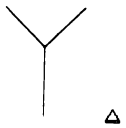
JOBB.IV 180 50 hatása

- Írjunk bal körívet rajzoló eljárást is!
- Írjunk olyan eljárásokat, amelyeknek hatására a Teknőc adott betűket kinagyítva rajzol a rajzmezőre! Például

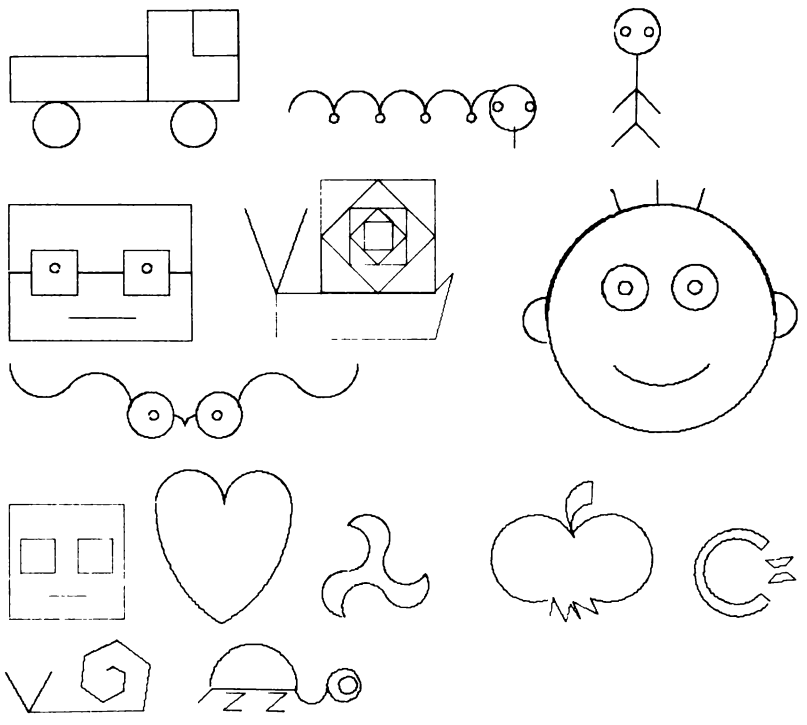


Útmutatás: Állapítsunk meg egy általános kezdő- és végpozíciót a tervezett betűnagyságnak megfelelően! Az egyes betűeljárások hatására a Teknőc a betű kirajzolása után vegye fel a következő betű kezdetének pozícióját!

Például:



- Próbáljunk megvalósítani a következő ábrák közül néhányat:



1.3.1. Hogyan fogjunk hozzá egy elképzelt ábra megrajzolásához?

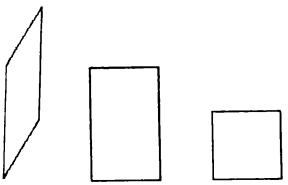
19 kérdésben és jó tanácsban összefoglaljuk a lépéseket.

Előkészítés

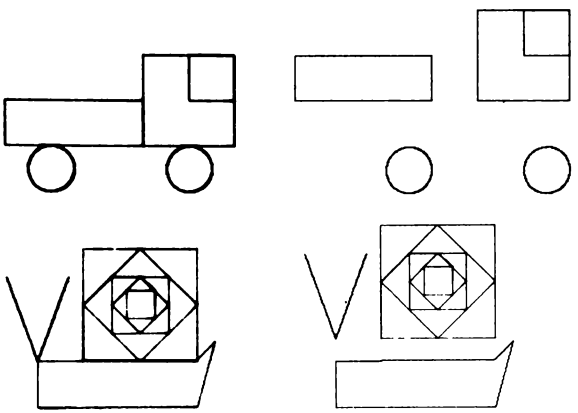
(1) *Rajzoljuk le elképzelésünket (lehetőleg kockás) papírra!*

Elgondolkodás

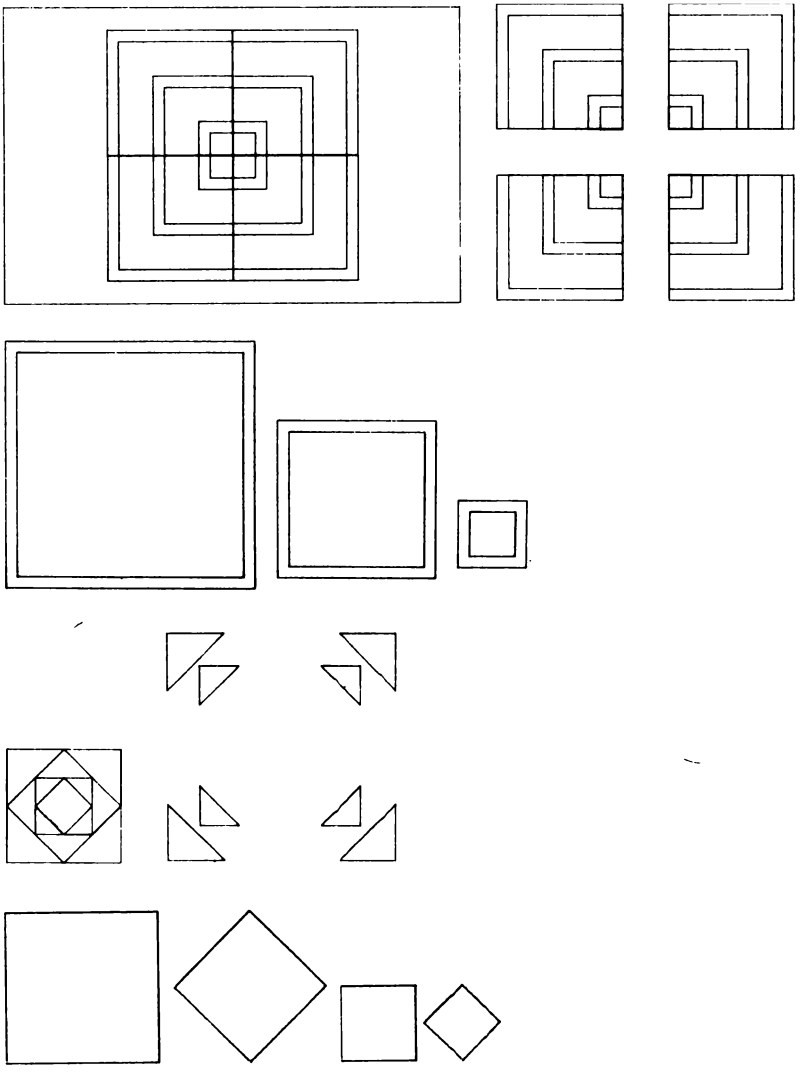
(2) *Nem készítettünk már ilyen ábrát? Esetleg a mostanitól kicsit eltérő formában? Nem tudnánk valahogy azt felhasználni? Kisebb módosítással nem tudnánk felhasználhatóvá tenni? Nem tudnánk felhasználni a módszerét?*



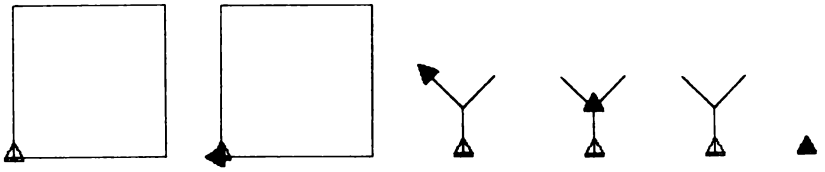
(3) *Próbáljuk az ábrát főbb részeire bontani!*



Ha nem jutunk semmire a felbontással, próbáljunk más szemmel tekinteni rá, és más szerkezetűre bontani!



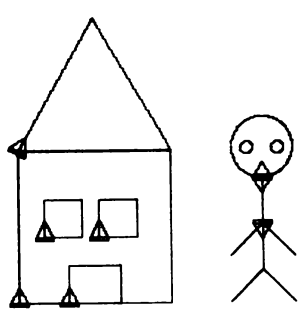
(4) *Határozzuk meg az egyes részek kezdő és végpontjában a Teknőc állapotát!*



Fontos azt is megvizsgálni, vajon a Teknőc tolla leeresztett vagy felemelt állapotból indul, ill. érkezik.

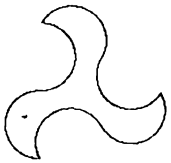
(5) *Gyakran hasznos az állapotinvariáns eljárások alkalmazása.*

Egy eljárás akkor állapotinvariáns, ha kezdő- és végállapota megegyezik (l. NEGYZET, HAROMSZOG ... stb.). Az így készített eljárások megkönnyíthetik a részek összerakásának folyamatát. Először csak a részek kiindulási pontjait kell megjelölnünk, míg az egyes részek előállításával nem törődünk. Csak később, az egyes részletek kidolgozásánál kell az állapotinvarianciákra ügyelnünk, hogy biztosak legyünk abban, hogy az ábra egészét nem rontottuk el. (Vagyis, nem kell ismernünk az egész ábra összerakásának mechanizmusát).

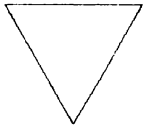


(6) *A bonyolult részeket először helyettesítsük állapotekvivalens eljárásokkal.*

Az azonos állapotátmenettel rendelkező eljárásokat nevezzük állapotekvivalenseknek. A helyettesítés nem változtatja meg az összerakás menetét.



állapotekvivalens

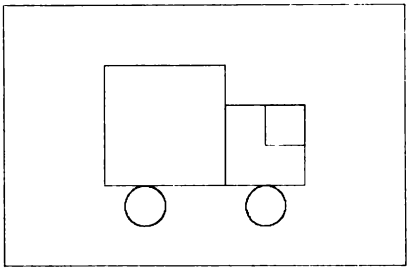


ahol

állapotekvivalens

Miután az ábra egészének így létrehozott eljárását kipróbáltuk és helyesnek bizonyult, ráérünk a helyettesített részlettel foglalkozni.

(7) *Használjuk fel a már meglevő általános eljárásokat!*



NEGYZET
KOR

(8) *Ha szükségünk van egy új eljárásra, írjuk meg!*

A Logo ugyan elég sok utasítást tartalmaz, de bárki szükségét érezheti olyan jól definiálható utasításoknak, amelyek megkönnyítenék a tevékenységek leírását. Az utasítások hatásuk szempontjából két csoportba oszthatók:

– azok, amelyeknek csak hatásuk van (pl. grafikus utasítások);
– azok, amelyek eredményt is adnak (pl. függvények).

A létrehozott felhasználói eljárások is ugyanígy csoportokba oszthatók, hiszen az utasítások egymásutánja belső felépítésük értelmében, meghatározott külső tulajdonságokat vált ki (képet hoz létre, eredményt ad ... stb.).

Az eredményszolgáltatást az OUTPUT utasítás alkalmazásával

lehet elérni, amelynek hatására az eljárás végrehajtása befejeződik, visszatér a hívás szintjére és eredményül adja az OUTPUT paraméterét.

```
TO ABS :X
 IF 0 > :X THEN OUTPUT :X * ( - 1 ) ELSE OUTPUT :X
END
```

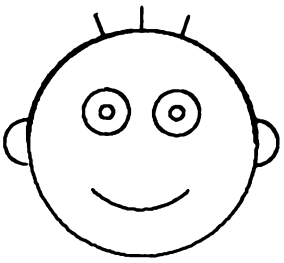
A kész eljárás neve úgy alkalmazható mint a beépített alaputasítások, így a funkciók pótlása nem probléma.

(9) *Írjunk általános eljárásokat!*

```
TO SOKSZOG :MERET :CSUCS
 REPEAT :CSUCS [FORWARD :MERET RIGHT 360 / :CSUCS]
END
```

Megvalósítás

(10) *Fogalmazzuk meg a részek összeillesztésének folyamatát!*



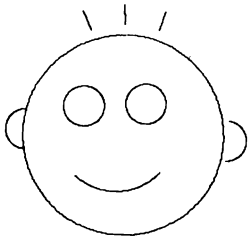
Az arc összetevőit két részre bonthatjuk, egy külső és egy belső részre. A külső rész alatt a fej körformáját, a füleket és a hajat érthetjük. A szemek és a száj legyenek a belső részek.

```
TO PETI
 KULSO
 BELSO
END

TO KULSO
 KOR 50
 FULEK
 HAJ
END

TO BELSO
 SZEMEK
 SZAJ
END
```

A fej körformáját akár a KOR1 eljárással is megrajzolhatjuk, de célszerűbb a KOR eljárás alkalmazása, mert pontosabban lehet így mérni a távolságokat a körök középpontjához képest.

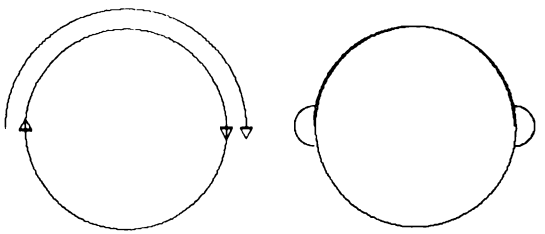
Hová helyezzük a füleket, a szemeket, a hajszálakat és a szájat? Keressük meg találomra a helyét?

Ha a Teknőcöt találomra mozgatjuk a becsült pozícióra, még kaphatunk tetszetős ábrát, de megfelelő tervezés nélkül nem mindig sikerül a szimmetriát elérni.



Tartományon belüli belső koordinátákkal érdemes próbálkoznunk.

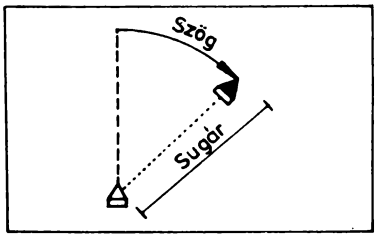
Egy megrajzolt ábra belső koordinátái pontosan meghatározhatók. A pontok elérésére az ábra eljárásai használhatók fel.



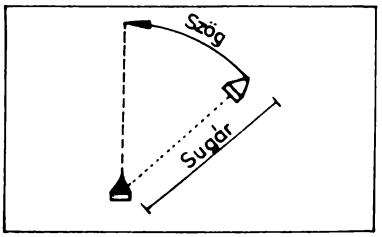
```
TO FULEK
 PENUP LEFT 90 FORWARD 50 LEFT 90 PENDOWN
 FUL
 RIGHT 180 JOBB.IV 180 50 RIGHT 180
 FUL
 PENUP RIGHT 90 BACK 50 LEFT 90 PENDOWN
END

TO FUL
 PENUP FORWARD 10 RIGHT 90 PENDOWN
 JOBB.IV 180 10
 PENUP RIGHT 90 FORWARD 10 PENDOWN
END.
```

A polárkoordinátákkal való megoldáshoz hasznos lehet a következő két segéd-eljárás

```
TO HELYEZ :SUGAR :SZOG
 PENUP RIGHT :SZOG FORWARD :SUGAR PENDOWN
END

TO VISSZA :SUGAR :SZOG
 PENUP BACK :SUGAR LEFT :SZOG PENDOWN
END
```



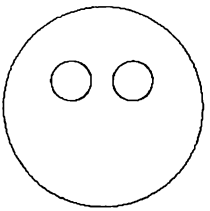
Helyez



Vissza

Így PETI szemeinek elhelyezését a következőképpen oldhatjuk meg:

```
TO SZEMEK
 HELYEZ 20 50
 SZEM
 VISSZA 20 50
 HELYEZ 20 ( - 50 )
 SZEM
 VISSZA 20 ( - 50 )
END

TO SZEM
 KOR 10
END
```



A maradék programrészek:

```
TO SZAJ
 PENUP RIGHT 135 FORWARD 30 RIGHT 90 PENDOWN
 JOBB.IV 90 30
END

TO HAJ
 LEFT 20
 REPEAT 3 [PENUP FORWARD 50 PENDOWN HAJSZ!
HL PENUP BACK 50 PENDOWN RIGHT 20 ]
 LEFT 40
END

TO HAJSZAL
 FORWARD 10
 BACK 10
END
```

Körszimmetrikus ábráknál a Descartes-koordinátákkal nem sokra megyünk, csak nehezítjük a rajzolást.

Az alakzatok elhelyezésére, hagyományos görberajzolásra vagy egyéb xy koordinátákkal jól megfogalmazható ábráknál azonban jól alkalmazhatók a SETX, SETY, SETXY utasítások

Miután egy pontot nem tudunk esetén egyszerűen kirajzolni, alkalmazzuk a PONT eljárást.

```
TO PONT
 FORWARD 1
 BACK 1
END
```

Ezt használjuk fel a következő, függvényrajzoló programban.

```
TO FUGGVENY
 SPLITSCREEN
 PRINT [MELYIK FUGGVENYT AKAROD KIRAJZOLNI?]
 MAKE "FGV FIRST REQUEST
 PRINT [MEKKORA LEGYEN AZ INTERVALLUM NAGYITASA?]
 MAKE "XNAGYITAS FIRST REQUEST
 PRINT [MEKKORA LEGYEN AZ ERTEKEK NAGYITASA?]
 MAKE "YNAGYITAS FIRST REQUEST
 PRINT [MEKKORA LEGYEN A LEPESKOZ?]
 MAKE "LEPESKOZ FIRST REQUEST
 PRINT [MI LEGYEN AZ INTERVALLUM FELSO HATARA?]
 MAKE "MAX FIRST REQUEST
 PRINT [MI LEGYEN AZ INTERVALLUM ALSO HATARA?]
 MAKE "MIN FIRST REQUEST
 XY.KOORD.TENG
 ABRAZOL :MIN
 PRINT [AKARSZ MASIK FUGGVENYT KIRAJZOLNI?]
 PRINT [IGEN VAGY NEM?]
 IF NOT ( FIRST REQUEST ) = "IGEN THEN STOP
 FUGGVENY
END
```

```
TO XY.KOORD.TENG
 .ASPECT 1
 NOWRAP
 DRAW
 FORWARD 100
 BACK 199
 FORWARD 99
 RIGHT 90
 FORWARD 159
 BACK 319
 FORWARD 160
 LEFT 90
END

TO ABRAZOL :X
 IF :X > :MAX THEN STOP
 PENUP
 SETX ( :X * :XNAGYITAS )
 SETY ( ERTEK :X ) * :YNAGYITAS
 PENDOWN
 PONT
 ABRAZOL :X + :LEPESKOZ
END

TO ERTEK :X
 OUTPUT RUN LIST :FGV :X
END
```

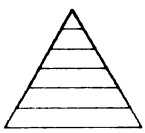
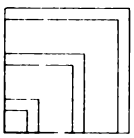
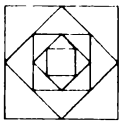
A FUGGVENY aktivizálása után választ kell adni a feltett kérdésekre (a függvény neve, az intervallum stb.). (A REQUEST működését l. a 265. oldalon.)
Például kirajzoltathatjuk a szinusz függvényt a [−180, 180] intervallumon a képernyőre illeszkedő nagyításban.

(11) *Alkalmazzunk általánosan kimondható tételeket!*

A Teknőc-geometria néhány alapvető tétele:

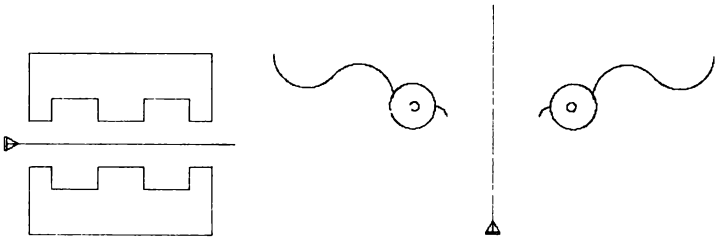
Hasonlósági tétel.

Ha a szögeket változatlanul hagyjuk, de egy szorzóval beszorozzuk a lépéseket, akkor a szorzó arányának megfelelő méretű alakzatot kapjuk.

Tükrözés tétele.

Ha a lépéseket változatlanul hagyjuk, de a fordulatok irányát ellentétesre változtatjuk, akkor a Teknőc kiinduló állapotán (pozíció és irány) átmenő egyenesre tükrözzük az alakzatot.



- Mi történik, ha csak a lépéseket váltjuk ellentétesre?
- Mi történik, ha a fordulatokat is és a lépéseket is ellentétesre váltjuk?

Útmutatás: A fordulatok megfordításának módjai:

– minden LEFT-et cseréljünk RIGHT-ra és minden RIGHT-ot cseréljünk LEFT-re;

– minden LEFT és RIGHT paramétert cseréljünk ellentétes előjelűre.

A lépések megfordításának módjai:

– minden FORWARD-ot cseréljünk BACK-re és minden BACK-et cseréljünk ~~RIGHT-re~~; FORWARD-ra;

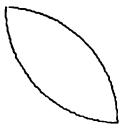
– minden FORWARD és ~~RIGHT~~ paramétert cseréljünk ellentétes előjelűre. BACK

Vigyázat! Nem minden eljárás viseli el a negatív paramétert (l. KOR1).

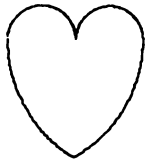
Teljes Teknőc-tétel

Ha a Teknőc akármilyen alakzatot bejárva pontosan ugyanabba az állapotba tér vissza, mint ahonnan kiindult, a fordulatok összege (pozitív, ha jobbra, negatív, ha balra fordult) 360°-ot vagy annak egész-szeresét adja.

```
TO SZIROM
 JOBB.IV 90 50
 RIGHT 90
 JOBB.IV 90 50
END
```



```
TO SZIV :MERET
 JOBB.IV 180 :MERET / 2
 RIGHT 180
 JOBB.IV 180 :MERET / 2
 JOBB.IV 60 :MERET * 2
 RIGHT 60
 JOBB.IV 60 :MERET * 2
END
```



(12) *Adjunk értelmes nevet eljárásainknak!*

Olyan nevet, amely jól tükrözi a feladatot. A legtöbb Logo változat (itt) gyakorlatilag korlátlan hosszú (max. 200 jelből álló) nevet engedélyez. Ne spóroljunk hát, de arra is vigyázzunk, nehogy olyan hosszú nevet adjunk, amely elírásokra ad alkalmat.

Például:

```
TO * :MERET
 JOBB.IV 180 :MERET / 2
 RIGHT 180
 JOBB.IV 180 :MERET / 2
 JOBB.IV 60 :MERET * 2
 RIGHT 60
 JOBB.IV 60 :MERET * 2
END

TO GORBE.HAROMSZOG :MERET
 REPEAT 3 [JOBB.IV 180 :MERET BAL.IV 180 !
:MERET RIGHT 120 ]
END
```

Ellenpélda:

%SZAMITAS$BANKIFEJEZVE

Kipróbálás

(13) *A kisebb próbálkozásokat parancsszinten végezzük el!*

Így rögtön látjuk utasításaink hatását. Egy összetett feladatnál azonban már hátrányt jelenthet, ha minden utasítást külön végeztetünk el, mert

- könnyen belezavarodunk a megoldás megetébe;
- hiba esetén előlről kell elkezdenünk a munkát;
- a kész megoldás nem reprodukálható.

(14) *Az összetett tevékenységeket foglaljuk eljárásba!*

Az eljárás aktivizálásával próbáljuk ki, mennyire sikerült elérni célunkat. Így újra meg újra elő tudjuk állítani a megalkotott képet, miközben az eljárás tartalmát megfelelően módosítjuk. Egy tevékenységsorozat azonban nem mindig érzékelteti a hozzá tartozó ábra felbontását, lényeges részeit. Ezért próbáljuk úgy megfogalmazni az eljárást, hogy minél jobban eligazodjunk benne:

– alkalmazzunk megjegyzéseket a fontosabb utasítások után.

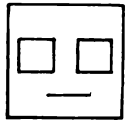
esetén a pontosvessző után minden jel megjegyzésnek számít, így végrehajtásra nem kerül sor.
Például:

```
TO FEJ
 RIGHT 90 ; SZAJ
 FORWARD 20
 RIGHT 90 ; ALL
 FORWARD 20
 RIGHT 90 ; FEJ
 FORWARD 50
 RIGHT 90
 FORWARD 60
 RIGHT 90
 FORWARD 50
 RIGHT 90 ; HOMLOK
 FORWARD 20
 LEFT 45 ; ORR
 FORWARD 10
 RIGHT 135
 FORWARD 5 ; SZEM
 PENUP
 FORWARD 20
 RIGHT 90
 FORWARD 10
END
```

Célszerű azokat a részeket is különválasztani, ahol a Teknőc felemelt tollal közlekedik, azoktól a részektől, amelyek nyomot is hagynak a rajzmezőn. Ezt könnyen el lehet érni egy egyszerű konvencióval: ha a PENUP és PENDOWN utasításokat teljesen kiírjuk, de a FORWARD, BACK, RIGHT, LEFT ... stb. utasításokat csak rövidítve használjuk, így nemcsak kevesebbet kell írnunk, de a két hosszabb utasítás zárójelként emelkedik ki a rövidek közül.

Például:

```
TO FERI
 NEGYZET 50
 PENUP RT 90 FD 5 LT 90 FD 20 PENDOWN
 NEGYZET 15
 PENUP RT 90 FD 25 LT 90 PENDOWN
 NEGYZET 15
 PENUP BK 10 RT 90 BK 15 PENDOWN
 FORWARD 20
 PENUP FD 15 RT 90 BK 10 PENDOWN
 HIDETURTLE
END
```



(A fejezetben kilistázott utasításokat teljes alakjukban használjuk, hogy az Olvasó könnyebben asszociálhasson az angol utasításnevek és a hozzájuk fűződő tevékenységek között.)

(15) *Kövessük a Teknőc mozgását a rajzmezőn látható állapotban!*

A SHOWTURTLE utasítás teszi láthatóvá a Teknőcöt, ha éppen nem az volt. Így könnyebben végigkísérhetjük tevékenykedését.

(16) *A fontos részeknél alkalmazzunk megszakításokat!*

Egy eljárás működése közben felfüggeszthető vagy megállítható.

– *Felfüggeszthető:* a <CTRL>_<Z> gombok lenyomásával.

Hatására az eljárás működése abbamarad és a szövegmezőre kiíródik a megállás pontjának szintszáma (milyen mélységű az eljárás). CONTINUE-val folytatható. A <CTRL>_<W> gombok lenyomására az eljárás működése abbamarad, de újra lenyomva folytatódik.

– *Megállítható:* a <CTRL>_<G> gombok lenyomásával. Hatására az eljárás működése abbamarad. Nem folytatható.

Ha nem akarjuk a reflexünkre bízni a felfüggesztés, ill. megállítás pillanatát, akkor építsük bele az eljárás szövegébe

– a felfüggesztést a PAUSE utasítással;
– a megállítást a TOPLEVEL utasítással.

A következő tevékenységeknek lehet értelme felfüggesztéskor:

– a Teknőc állapotának vizsgálata;
– egyéb tényezők lekérdezése;
– a változók tartalmának lekérdezése;
– módosítások elvégzése.

(17) *Kövessük nyomon a hibás részeket!*

A TRACE és NOTRACE utasítások közé foglalhatjuk a nyomon követendő utasítássorozatot.

Például:

```
TO EMBER
 KOR 20
 PENUP
 RIGHT 90
 FORWARD 20
 RIGHT 90
 FORWARD 20
 PENDOWN
 TRACE
 Y
 NOTRACE
 Y
END

TO Y
 FORWARD 30
 RIGHT 45
 FORWARD 30
 BACK 30
 LEFT 90
 FORWARD 30
 BACK 30
 RIGHT 45
END

TO KOR :X
 REPEAT 360 [FORWARD :X * 0.0174 RIGHT 1]
END
```

Az EMBER eljárás mindaddig normálisan működik, amíg a TRACE utasításig el nem jut. Utána csak lépésenként hajtja végre az egyes utasításokat, mindig leírva a következő végrehajtandó utasítást.

```
TRACING ON
  Y
 EXECUTING Y
    FORWARD 30
    RIGHT 45
    FORWARD 30
    BACK 30
    LEFT 90
    FORWARD 30
    BACK 30
    RIGHT 45
 ENDING Y
  NOTRACE
TRACING OFF
```

Bármely billentyű lenyomására (kivéve a <CTRL>_<Z>, ill. <CTRL>_ _<G>) végrehajtja és leírja a következő sort. Érdemes kevés utasítást (lehetőleg egyet) írni egy sorba, hogy hatásosabb legyen a nyomkövetés. Ilyenkor is lehet felfüggesztést alkalmazni.

(18) *Személyesítsük meg a Teknőcöt!*

Ha már mindent kipróbáltunk eljárásunk működésének vizsgálatára, de nem jöttünk rá, hogy hol követtünk el hibát, ill. miért viselkedik hibásan a Teknőc, akkor álljunk fel és képzeljük magunkat a helyébe.
Próbáljuk meg teljesíteni a Teknőcnek szánt utasításokat.

(19) *A megfelelően működő programjainkat mentsük ki!*

– A kész eljárásokat SAVE utasítással vihetjük háttértárra. A rajzmező tartalmát a SAVEPICT utasítás segítségével tudjuk kimenteni.

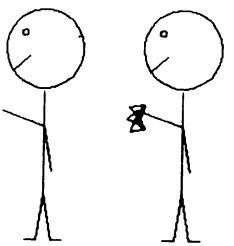
Rajzgép vagy nyomtató segítségével az ábrákat papírra is vethetjük, de erre terjedelmi okokból nem térünk ki.

1.4. Grafikai kellékek

1.4.1. Animáció

Kisebb-nagyobb trukkök már elegendőek az egyszerű animáció létrehozására (például a Teknőc forgatásával).
Rajzoljunk egy kis emberkét! Az emberke megrajzolása azzal végződjön, hogy a Teknőc a kéz végére kerül.

```
TO INTEGET
 REPEAT 15 [RIGHT 5]
 REPEAT 15 [LEFT 5]
 INTEGET
END
```



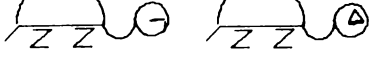
Ekkor aktivizáljuk az INTEGET eljárást, amelynek hatására az emberke addig integet, amíg drasztikusan (<CTRL> _ <G>-vel) le nem állítjuk.

Az integetést gyorsítani, ill. lassítani lehet a RIGHT és LEFT paramétereinek megváltoztatásával (úgy, hogy a REPEAT ismétléseinek számát megfelelően igazítjuk).

A Teknőc eltüntetésével pl. pislogást animálhatunk:

```
TO PISLOGO.TEKNOC
 RIGHT 45 FAROK
 RIGHT 45 HAT
 LABAK
 RIGHT 90 NYAK
 FEJ1
END

TO FEJ1
 GORBE 2 8 360
 PENUP RIGHT 90 FORWARD 18 LEFT 90 PENDOWN
 RIGHT 90 BACK 5 FORWARD 10 BACK 5 ; SZEM
 PISLOG
END

TO PISLOG
 VARJ 1000
 HIDETURTLE
 VARJ 1000
 SHOWTURTLE
 PISLOG
END

TO VARJ :IDO
 REPEAT :IDO []
END
```

A PISLOGO.TEKNOC programban felhasználtuk a már meglevő TEKNOC programunkat (l. 300. old.). A Teknőc megjelenése és eltűnése a pislogás hatását kelti. A VARJ eljárással lehet lassítani a folyamatot.



Az állandó mozgás szimulálását a *Teknőc folytonos előrehaladásával* lehet elérni:

```
TO MOZOG
 FORWARD 5
 MOZOG
END
```

A Teknőc tollának felemelése után (PENUP) aktivizáljuk a MOZOG eljárást. A FORWARD paraméterének nagyságától függ a gyorsaság. Ha ezt a mozgást gombnyomásokkal befolyásolni akarjuk, hasznos lehet a következő eljárás:

```
TO PARANCS
 MAKE "PAR BEOLVAS
 IF :PAR = "J THEN RT 30
 IF :PAR = "B THEN LT 30
 IF :PAR = "E THEN LOKES
```

```
TO BEOLVAS
 IF RC? THEN OUTPUT RC
 OUTPUT "
END
```

```
 IF :PAR = "F THEN PENUP
 IF :PAR = "L THEN PENDOWN
 IF :PAR = "+ THEN MAKE "ERO :ERO + 1
 IF :PAR = "- THEN MAKE "ERO :ERO - 1
END
```

Itt az eredmény a lenyomott gomb jele (ha nem történt gombnyomás, úgy üres jel), amellyel különböző hatásokat érhetünk el.
A mozgás a Descartes-koordináták alkalmazásával (SETXY) gyorsítható:

```
TO LOKES
 MAKE "VX :VX + :ERO * ( SIN HEADING )
 MAKE "VY :VY + :ERO * ( COS HEADING )
END

TO MOZGAS
 SETXY ( XCOR + :VX ) ( YCOR + :VY )
END
```

A folyamatosságot a következő eljárás biztosítja:

```
TO DINA
 MOZGAS
 PARANCS
 DINA
END
```

A folyamat inicializálásának eljárása pedig a következő:

```
TO DINAMIKUSTEKNOC
 DRAW
 MAKE "VX 0
 MAKE "VY 0
 MAKE "ERO 1
 DINA
END
```

A DINAMIKUSTEKNOC aktivizálásával egy, a Newton-törvényeknek engedelmeskedő Teknőcöt kapunk, amelynek mozgását lökésekkel befolyásolhatjuk. A Teknőc irányát a lökés iránya és az eredeti irány együttese szabja meg. (J: jobbra forgatja, B: balra forgatja.) A lökés erősségét is befolyásolhatjuk interaktív módon (+: növeli, –: csökkenti).

- Figyeljük meg, milyen irányban tér ki a Teknőc egy-egy lökés hatására.
- Manőverezzünk a Teknőccel. Próbáljuk meg körpályára bírni.

Rajzolással bonyolultabb animáció is elérhető, ha az igazi animációs filmek készítéséből merítünk. Ezek a hatások a fázisrajzolás és a törlés folytonos ismétlésével érhetők el. Természetesen a fázisrajzok lépésenként módosulnak. Mozgassunk egy négyzetet a képernyőn! Alkalmazzuk az ismert négyzetrajzoló eljárást!

```
TO NEGYZET :MERET
 REPEAT 4 [FORWARD :MERET RIGHT 90]
END
```

A fázisrajzolás a négyzet megrajzolásából és a toll áthelyezéséből áll, majd ezt követi a törlés.

```
TO N.C1
 NEGYZET 10
 PENUP
 FORWARD 10
 PENDOWN
 CLEARSCREEN
END
```

A folyamat ismétlésének eljárása:

```
TO NEGYZET.CSUSZ1
 REPEAT 10 [N.C1]
END
```

A fázis megrajzolása után érdemes várakozni egy ideig a letörlés előtt, hogy a megrajzolt alak tovább legyen látható, mint a törlés és újrarajzolás ideje. Természetesen a mikrogépek nem alkalmasak az ilyen fajta animáció gyors lebonyolítására, de alkalmazhatunk néhány trükköt a folyamat gyorsítása érdekében:

– a megrajzolandó (módosuló) alak kis méretű és lehetőleg egyszerű legyen;

– az eltüntetett Teknőc (HIDETURTLE) gyorsabban rajzol;

– a Teknőc mozgatása gyorsabb, ha a FORWARD, BACK, RIGHT, LEFT alkalmazása helyett a SETXY-t használjuk;

– a fázis törlését radírral való átrajzolással oldjuk meg.

```
TO N.C2
 PENCOLOR 1
 NEGYZET 10
 VARJ
 PENERASE
 NEGYZET 10
 PENUP
 FORWARD 10
 PENDOWN
END

TO VARJ
 REPEAT 300 []
END
```

Az animáció variálására lássunk néhány tippet:

– az alak mozgása görbe mentén

```
TO N.C3          PENUP
 PENCOLOR 1      RIGHT 30
 NEGYZET 10      FORWARD 10
 VARJ            PENDOWN
 PENERASE       END
 NEGYZET 10
```

– az alak méretének módosulása mint távolodás és közeledés

```
TO NEGYZET.CSUSZ2
 MAKE "MERET 5
 REPEAT 10 [N.C4 MAKE "MERET :MERET + 5]
END

TO N.C4
 PENCOLOR 1
 NEGYZET :MERET
 VARJ
 PENERASE
 NEGYZET :MERET
 PENUP
 RIGHT 30
 FORWARD 10
 PENDOWN
END
```

– az alak gyorsított, ill. lassított mozgása

```
TO NEGYZET.CSUSZ3
 MAKE "GYORS 1
 REPEAT 10 [N.C5 MAKE "GYORS :GYORS + 5]
END

TO N.C5
 PENCOLOR 1
 NEGYZET 10
 VARJ
 PENERASE
 NEGYZET 10
 PENUP
 FORWARD :GYORS
 PENDOWN
END
```

- Rajzoljunk órát és szimuláljuk a mutató járását!

A leghatékonyabb módszer az animálásra a *sprite-ok* (vagyis mozgatható alakzatok) alkalmazása.

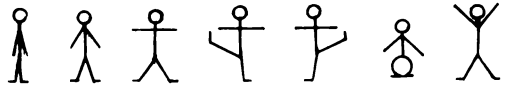
Az eddig tárgyalt rajzolásos animáció bizonyos fajtái gyorsíthatók, ha pl. az egyik sprite (Teknőc) rajzol, a másik pedig töröl.

```
TO HERNYO
 TELL 1 HOME
 TELL 0 DRAW HIDETURTLE FORWARD 20
 MOZGAS
END

TO MOZGAS
 TELL 0 FORWARD 5
 VARJ
 TELL 1 FORWARD 5
 VARJ
 MOZGAS
END
```

A sprite-okkal végzett speciális animáció két részre bontható:

a) Az alakzat variálása

Egy alak változását a sprite-alakzatok cserélgetésével érhetjük el. A különböző sprite-alakzatokat a mozgásfázisnak megfelelően definiáljuk. A sprite-hoz rendelt alakzat számának módosításával (SETSHAPE) hozhatjuk létre a mozgást.



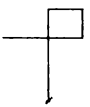
b) Alakzat helyváltoztatása

Mivel minden sprite (még a Teknőc is mint 0. sprite) ugyanazon utasítások segítségével (nevezetesen Teknőc-utasításokkal) mozgatható, csupán a megfelelő sorszámú sprite-ot kell aktívvá tenni (TELL), majd a kívánt Teknőc-utasítással mozgásba lendíteni.

1.4.2. Véletlenszámok alkalmazása

A RANDOM utasítás eredménye egy véletlen (egész) szám, amely paraméterétől függő intervallumba esik. Az intervallum alsó határán 0, felső határa pedig a (paraméterérték-1). Így a RANDOM 8 eredménye 0 és 7 közötti egész szám lehet.

```
REPEAT 4 [FORWARD RANDOM 50 RIGHT 90]
```



Különböző effektusokat lehet elérni csupán a háttérszín véletlenszerű váltogatásával:

```
TO VILLAN
 BACKGROUND RANDOM 16
 VARJ
 VILLAN
END
```

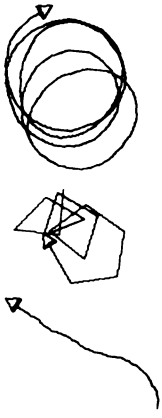
Tetszetősebb effektus érhető el azonban, ha valamilyen animációs fázissal, pl. rajzolással lassítjuk a folyamatot:

```
TO VILLAN
 BACKGROUND RANDOM 16
 V.RAJZ
 VILLAN
END
```

```
TO V.RAJZ
 PENCOLOR RANDOM 16
 REPEAT 4 [FORWARD RANDOM 50 RIGHT RANDOM 360]
END
```

A RANDOM értelmezési tartományának intervallumát módosíthatjuk. A különböző intervallumok hatására más-más fajta mozgást érhetünk el: röpködést, kóválygást, imbolygást ... stb.

```
TO ROPKOD
 FORWARD 20
 RIGHT ( 60 + RANDOM 90 )
 ROPKOD
END

TO KOVALYOG
 FORWARD 2
 RIGHT RANDOM 10
 KOVALYOG
END

TO IMBOLYOG
 FORWARD 1
 RIGHT ( - 10 + RANDOM 20 )
 IMBOLYOG
END
```



1.4.3. Színezés

Nem foglalkoztunk még a színezéssel. Ennek az az oka, hogy a tárgyalt Logo nyelvjárások egyikébe sincs beépítve az alakzatok kiszínezésére valamilyen eljárás. A sokféle algoritmus közül egy igen egyszerűt mutatunk be. Képzeljük el a következő színezési lépéseket!

– Szúrjunk egy gombostűt a rajzlapba!

– Feszítsünk ki egy befőttes gumit a gombostű és az ábrát rajzoló ceruza közé!

– Rajzoljuk meg az ábrát!

Ha előzőleg a gumit festékbe mártottuk volna, akkor a rajzolás egyben színezés is volna.

Alkalmazzuk ezt a módszert az eljárás megfogalmazásánál. A FORWARD utasítás helyett pl. olyan eljárást írunk, amely úgy teszi meg az adott távolságot előre, hogy minden kis lépés után vonalat húz egy *belső pontba*, majd visszatér és folytatja útját.

```
TO SZINEZVE.ELORE :TAVOLSAG
 IF :TAVOLSAG = 0 THEN STOP
 MAKE "XPOZICIO XCOR
 MAKE "YPOZICIO YCOR
 SETXY :BELSO.X :BELSO.Y
 SETXY :XPOZICIO :YPOZICIO
 FORWARD 1
 SZINEZVE.ELORE :TAVOLSAG - 1
END
```

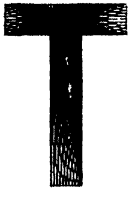
Ez már alkalmas is arra, hogy különböző alakzatok rajzolását a SZINEZVE. ELORE utasítással fogalmazzuk meg, csak a rajzolás előtt meg kell még határozni a *belső pont* koordinátáit. Magát a belső pont megválasztását mindig az alakzattól függően, körültekintően kell elvégezni.

```
TO NEGYZET :MERET
 MAKE "BELSO.X XCOR
 MAKE "BELSO.Y YCOR
 REPEAT 4 [SZINEZVE.ELORE :MERET RT 90]
END
```

vagy

```
TO NEGYZET :MERET
 MAKE "BELSO.X XCOR + :MERET / 2
 MAKE "BELSO.Y YCOR + :MERET / 2
 REPEAT 4 [SZINEZVE.ELORE :MERET RT 90]
END
```

```
TO T
 MAKE "BELSO.X XCOR
 MAKE "BELSO.Y YCOR + 110
 SZINEZVE.ELORE 100
 LEFT 90
 SZINEZVE.ELORE 30
 RIGHT 90
 SZINEZVE.ELORE 20
 RIGHT 90
 SZINEZVE.ELORE 80
 RIGHT 90
 SZINEZVE.ELORE 20
 RIGHT 90
 SZINEZVE.ELORE 30
 LEFT 90
 SZINEZVE.ELORE 100
 RIGHT 90
 SZINEZVE.ELORE 20
 RIGHT 90
END
```



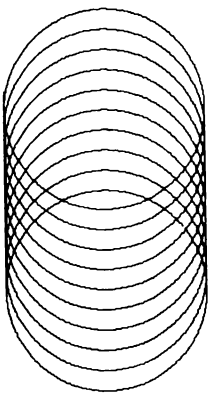
1.5. A Teknőc-grafika hatékonysága

Ebben a szakaszban felsorakoztatjuk a Teknőc-grafika színét, javát. Az eddigiekben megismerkedtünk a Teknőc-grafika elemeivel és meglevő rajzokat utánoztunk. Most azonban a Teknőc adottságaira épített ábrákat készítünk. Előre pillantva az ábrákra, meggyőződhet az Olvasó, hogy ténylegesen létezik ilyen adottság, amely a következetes ismételgetések szemgyönyörködtető tulajdonságában rejlik.

1.5.1. Ismétlések

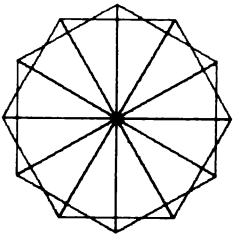
Az ismételgetéshez először is meg kell állapodjunk egy elemben, amit ismételgetni akarunk. Ha pl. egy állapotátlátszó ábrát ismételgetünk (végpont-kezdőpont), akkor a puszta ismétlés nem nyújt semmi újat, tehát valami kis módosításra van szükség. A Teknőc mozgásának alapvető két elemét iktatjuk be módosításként a FORWARD–BACKWARD-eltolást és a RIGHT–LEFT-forgatást. Legyen az első az eltolás.

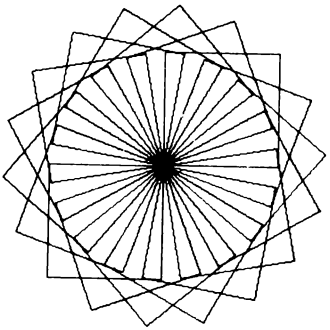
```
REPEAT 10 [KOR 50 FORWARD 10]
```

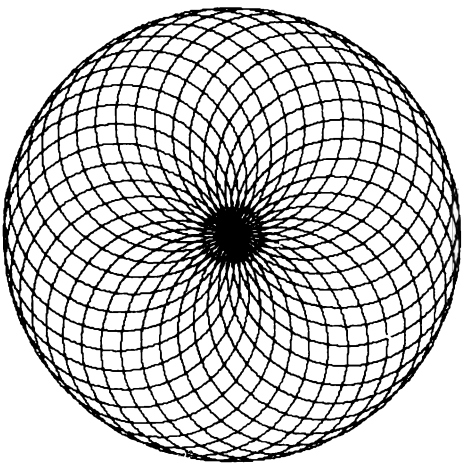


Ez még egy kicsit unalmas, úgyhogy eltolás helyett iktassunk be forgatást. Tehát vegyünk egy tetszőleges állapotátlátszó ábrát (a legegyszerűbbeket: háromszög, négyzet, kör) és forgassuk el 360° valahányad részével. Az ismétlést pedig annyiszor kívánjuk meg, hogy éppen bejárja a Teknőc a teljes 360°-ot.

```
REPEAT 12 [RIGHT 30 HAROMSZOG 50]
REPEAT 18 [RIGHT 20 NEGYZET 50]
REPEAT 36 [RIGHT 10 KOR 50]
```

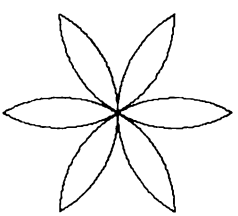




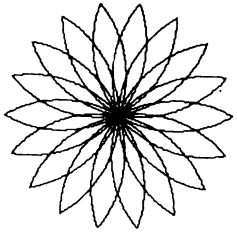


Elvégezhetjük ezt más alakzatokkal is:

```
TO VIRAG
 REPEAT 6 [RIGHT 60 SZIROM]
END

TO SZIROM
 JOBB.IV 60 50
 RIGHT 120
 JOBB.IV 60 50
 RIGHT 120
END
```



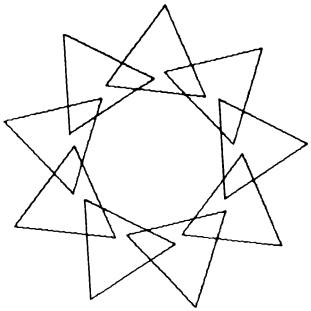
Ha a szirmokat sűrűbbre akarjuk, akkor egy szirom megrajzolása után kisebb szöggel fordítsuk el a Teknőcöt, pl. így:

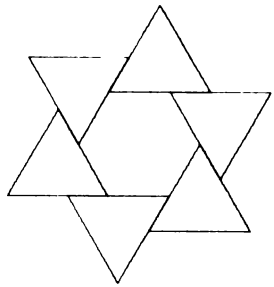


```
TO VIRAG
 REPEAT 18 [RIGHT 20 SZIROM]
END
```

Kipróbálhatjuk az ismétlés, eltolás, forgatás egymásutánját is!

REPEAT 9 [PENUP RIGHT 40 FORWARD 30 PENDOWN HAROMSZOG 50]
REPEAT 6 [PENUP RIGHT 60 FORWARD 30 PENDOWN HAROMSZOG 50]





(A PENUP eltünteti az eltolás vonalát.)

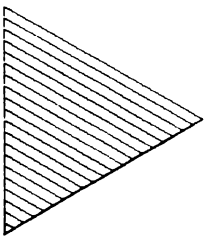
Eddig az ismétlés alapeleme változatlan volt. Most módosítsunk ezen is! Az alakzat paraméterét minden egyes lépésnél egy alkalmasan választott értékkel növelve vagy csökkentve rajzoljuk újra az ábrát. Az így létrehozott eljárásnak sosincs vége, ezért gondoskodnunk kell a leállításáról.

```
TO KEP1 :MERET
 IF :MERET > 50 THEN STOP
 KOR :MERET
 KEP1 :MERET + 5
END
```

A KEP1 program jól demonstrálja, hogy a Logo-ban milyen kézenfekvő a rekurzió. A programot meghívva, hacsak a méret nem több 50-nél, egy mé-

ret sugarú kör jelenik meg, majd a következő lépésben a program újra hívja önmegát méret+5 paraméterrel. Ebben az esetben tehát lokálisan módosítottuk a paramétert és nem volt szükség a MAKE utasításra. A leállási feltétel helyét gondosan kell megváltoztatni, ha ugyanis a rekurzív utasítás után írjuk, akkor sosem kerül sorra.

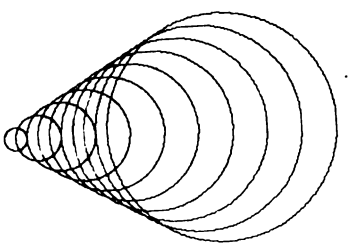
- Próbáljuk ki a KEP1 programot és készítsünk egy hasonlót KEP2 néven, amelyik hasonló háromszögek egymásutánját állítja elő.



```
TO KEP2 :MERET
 IF :MERET > 100 THEN STOP
 HAROMSZOG :MERET
 KEP2 :MERET + 5
END
```

Folytassuk az ábrakészítést az eltolás beiktatásával. Tehát: ismétlés, paraméterváltoztatás és eltolás egymásutánjait.

```
TO KEP :MERET
 IF :MERET > 50 THEN STOP
 KOR :MERET
 PENUP
 RIGHT 90
 FORWARD 5
 LEFT 90
 PENDOWN
 KEP :MERET + 5
END
```

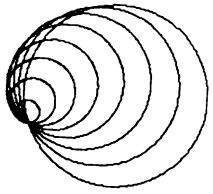


- Próbáljuk ki a programot a RIGHT 90, LEFT 90 kihagyásával is, így rögtön kitűnik, hogy mi a jelentősége.
- Milyen irányban forduljon a Teknőc jobbra, ill. balra, ha a növekvő körök középpontjainak összekötő egyenese délkeletre mutasson?

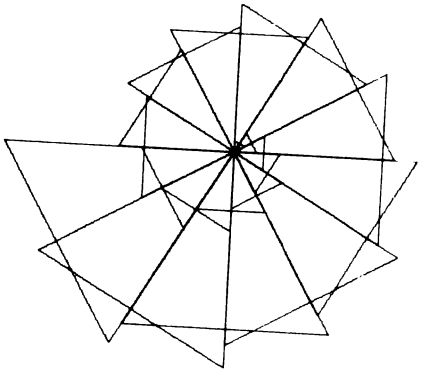
A következő lépés megint a forgatás.

A következő programokban elég önkényesen választottuk meg az adatokat, hogy különböző csigavonalak rajzolódjanak ki. Egyébként minden egyes programban érdemes kísérletezni a különböző adatokkal.

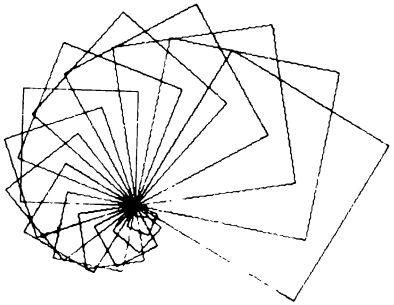
```
TO KEP1 :MERET
 IF :MERET > 40 THEN STOP
 KOR :MERET
 RIGHT 10
 KEP1 :MERET + 5
END
```



```
TO KEP2 :MERET
 IF :MERET > 100 THEN STOP
 HAROMSZOG :MERET
 RIGHT 30
 KEP2 :MERET + 5
END
```



```
TO KEP3 :MERET
 IF :MERET > 80 THEN STOP
 NEGYZET :MERET
 RIGHT 20
 KEP3 :MERET + 5
END
```



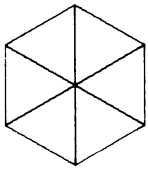
1.5.2. Síkkitöltő ábrák

Parkettázásnak vagy csempézésnek szokták (találóan!) nevezni azokat a kérdéseket, feladatokat, amelyeket a sík azonos (vagy véges sokféle) elemekből való lefedése vet fel. Most csak a szimmetrikus poligonokra szorítkozunk.

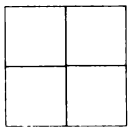
- Felvethető pl., hogy milyen szabályos poligonokkal lehet a hézagmentes kitöltést megoldani?

A kísérletezéshez adunk néhány ötletet:

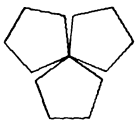
REPEAT 6 [RIGHT 60 HAROMSZOG 50]



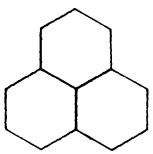
REPEAT 4 [RIGHT 90 NEGYZET 50]



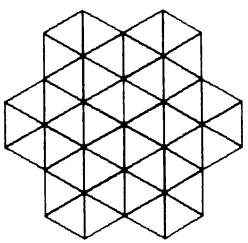
REPEAT 3 [RIGHT 120 OTSZOG 50]



REPEAT 3 [RIGHT 120 HATSZOG 50]



• Próbáljuk meg a képernyősíkot szabályos háromszögekkel lefedni, ahogy ábránkon bemutatjuk.

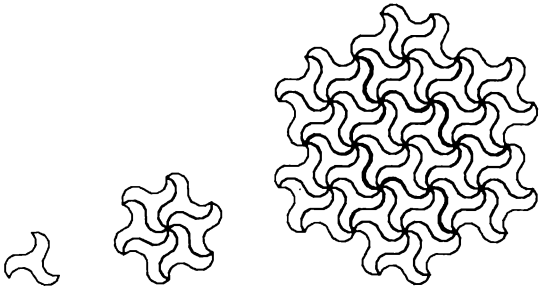


Egy ilyen programmal két legyet ütünk egy csapásra, hiszen a szabályos hatszögekkel való lefedést is megoldottuk. Egy egyszerű és esztétikus irányban folytatva a síkkitöltést, helyettesítsük a háromszög oldalait egy görbe S vonallal.

```
TO S :FOK :MERET
 REPEAT 2 [BAL.IV :FOK :MERET JOBB.IV :FOK :MERET]
END

TO GORBE.HAROMSZOG :MERET
 REPEAT 3 [S 180 :MERET / 4 RIGHT 120]
END

TO CS12 :MERET
 REPEAT 6 [GORBE.HAROMSZOG :MERET RIGHT 60]
END
```

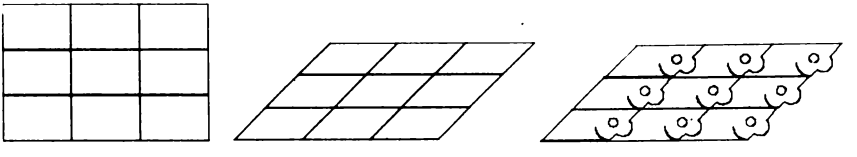
Mivel az S rajzolása így nagyon lassú, ezért egy alig látható csaláshoz folyamodunk:

```
TO S
 REPEAT 12 [FORWARD 2 RIGHT 15]
 REPEAT 12 [LEFT 15 FORWARD 2]
END
```



- További ötletek sora áll nyitva ebben az irányban is az Olvasó előtt!

Mi pedig bemutatunk még egy lehetséges módját a parkettázásnak. Téglalapokból kiindulva először paralelogrammákat készítünk, majd ezek egyik oldalát tetszőlegesen csinosítjuk. (Az összeillesztés itt nem elforgatással történik.)



E legutóbbi példa már a mintázáshoz vagy mozaikozáshoz állt közelebb. Itt nem elsősorban az alakzattal való síkkitöltés, hanem adott parkettázás szimmetrikus díszítése a feladat.

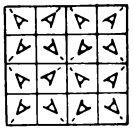
A mozaikozás egy igen látványos megoldása a következő: tekintsünk egy megfelelő nagyságú négyzetet, és benne egy tetszőleges ábrát.

a)

Osszuk fel a négyzetet 4 egyenlő részre és illesszük a lekicsinyített ábrát az egyikbe, forgatásait a többibe!

b)

Folytassuk ugyanígy az eljárást: újabb felosztás, kicsinyítés, beillesztés.



c)

Az eljárást a következő program adja meg:

```
TO ILLESZTES :X :ELJARAS :M
 FORWARD :M / 2
 LEFT 90
 FORWARD :M / 2 RIGHT 90
 REPEAT 4 [OSSZERAK :X :ELJARAS :M]
 LEFT 90
 BACK :M / 2
 RIGHT 90
 BACK :M / 2
END
```

```
TO OSSZERAK :FORGATAS :ELJARAS :MERET
 FORWARD :MERET / 2 LEFT 90
 REPEAT :FORGATAS [FORWARD :MERET / 2 LEFT 90]
 MINTARAJZ :ELJARAS :MERET / 2
 REPEAT :FORGATAS [RIGHT 90 BACK :MERET / 2]
 RIGHT 90 BACK :MERET / 2
 RIGHT 90
END

TO MINTARAJZ :ELJARAS :INPUT
 RUN SENTENCE :ELJARAS :INPUT
END
```

AZ ILLESZTES eljárás hozza létre az előbb említett folyamatot. Három paraméter játszik benne kulcsszerepet

X: az alapábra forgatásainak száma;

ELJARAS: az alapábra megrajzolásának eljárásneve;

M: az összkép mérete.

Az eljárás meghívása feltételezi, hogy a Teknőc pillanatnyi pozíciója az összkép jobb alsó csücske. A csempék összerakását mindig a négy csempe közepétől kezdi, így az eljárás eleje középre, vége pedig vissza helyezi a Teknőcöt.

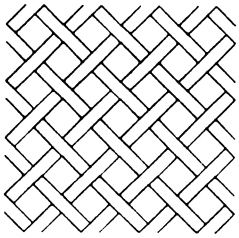
Az ILLESZTES középső része végzi el a tulajdonképpeni csempeösszerakást az OSSZERAK eljárás meghívásával. Három paramétere átveszi az ILLESZTES paramétereinek értékét. Az OSSZERAK eljárás struktúrája is hasonló az ILLESZTES-éhez: elforgatás, mintarajzolás, visszaforgatás. Így a minta rajzolását a MINTARAJZ eljárás végzi el, amelynek ELJARAS paramétertartalma változatlan marad, INPUT paramétere viszont a megfelelő kicsinyített mérettel dolgozik. A MINTARAJZ egyetlen feladata az alapminta megfelelő méretparaméterrel való végrehajtása.

A következő fokozathoz már csupán egyetlen lépés beiktatása szükséges. Illesztésünk ne egy alapábrával dolgozzon, hanem egy már összeillesztett ábrával. Azaz illesztésünk 2. paramétere ne a minta, hanem egy másik illesztés legyen! Ezt a beágyazást többször is elvégezhetjük.

Válasszunk most ILLESZTES-hez egy tetszőleges ábrát.

```
TO MINI :M
 FORWARD 2 * :M / 3
 LEFT 135
 PENDOWN
 FORWARD 2 * ( SQRT 2 ) * :M / 3
 RIGHT 90
 FORWARD ( SQRT 2 ) * :M / 3
 RIGHT 90
 FORWARD 2 * ( SQRT 2 ) * :M / 3
 PENUP
 RIGHT 45
 FORWARD :M / 3
 RIGHT 90
 FORWARD :M
 RIGHT 180
END
```

Aktivizálva a következő illesztéssorozatot, kapjuk a mozaikszerű ábrát.

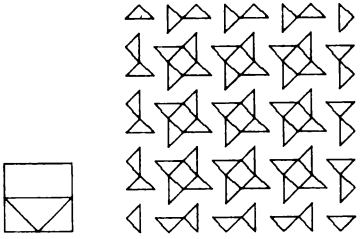


```
DRAW
PENUP
SETXY 100 (-100)
ILLESZTES 3 [ILLESZTES 3 [ILLESZTES 3 [MIN 1]]] 200
```

Az első három utasítás csupán megadja az ábra rajzolásának kiindulási helyét, amely mindig a jobb alsó sarok.

- Figyeljük meg, hogyan módosul a teljes kép a forgatások számának megváltoztatásával (ILLESZTES első paramétere).
- A legnagyobb fantázia megfelelő alapábra kitalálásában rejlik. A feladat lényege: olyan ábrát találni, amelynek megfelelő illesztései valamilyen tetszetős mintát eredményeznek.*

Például:



* A mozaikgrafika legnagyobb mestere M. C. Escher (1898–1972) holland grafikus volt. Grafikái alapján jó néhány ötletet meríthet az Olvasó. Úgy mondják egyébként, hogy Escher a matematikusok kedvenc festője, mert számos matematikai feladvány rejtőzik képeiben (egyköpenyű testek, több dimenziós terek síkbeli szimulációja stb.). Az amerikai matematikus Hofstetter: Bach-Gödel-Escher c. könyvének már a címe is erről árulkodik.

1.5.3. Csillagok és spirálok

Tárgyalásunkban eddig csak állapotátlátszó alakzatokról volt szó. Most rátérünk a nem állapotátlátszó alakzatokra, vagyis a nyílt görbékre.

Először is megállapíthatjuk, hogy minden szabályos poligon szakaszok ismételt elforgatásából tevődik össze.

Például az ötszög:

```
REPEAT 5 [RIGHT 72 FORWARD 50].
```

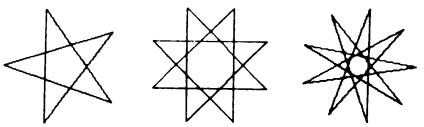
Ebből könnyen általánosíthatunk tetszőleges méretű és csúcsú szabályos poligon előállítására.

```
REPEAT :CSUCS [RIGHT 360/ :CSUCS FORWARD :MERET].
```

Az eddigi poligonok mind konvexek voltak. Konkávokká válnak, vagyis egyszerűbben, csillag formájúak lesznek, ha a csúcsszám és a forgásszög szorzata 360° többszöröse.

Például:

```
REPEAT 5 [RIGHT 144 FORWARD 50]
REPEAT 8 [RIGHT 225 FORWARD 50]
REPEAT 9 [RIGHT 200 FORWARD 50].
```



• Írjuk fel az általános csillagrajzoló eljárást! Természetesen a 360° többszörösét *alkalmasan* kell megválasztani, különben meglepetés érhet. Mint pl.

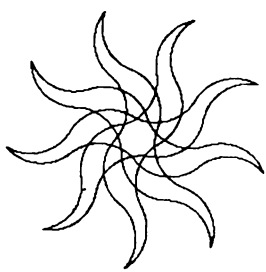
```
REPEAT 6 [RIGHT 360 * 2/6 FORWARD 50]
```

esetén.

• Milyen kikötést kell tegyünk 360° szorzójára, hogy egyértelműen csillagot kapjunk eredményül?

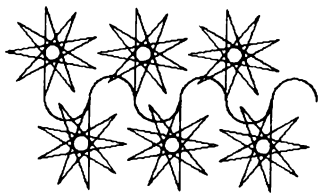
Ismét megtehetjük, hogy az egyenes szakaszokat görbével helyettesítjük.

```
TO CSILLAG
 REPEAT 9 [RIGHT 160 S]
END

TO S
 REPEAT 2 [BAL.IV 90 20 JOBB.IV 90 20]
END
```



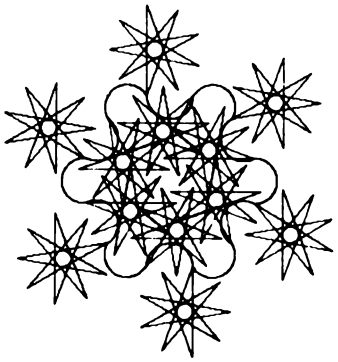
Tovább variálhatjuk a szakaszokat egyre bonyolultabb minták hozzáadásával. Megtoldhatjuk a nem állapotátlátszó görbét egy állapotátlátszóval (amikor is az eredmény nem állapotátlátszó görbe marad).

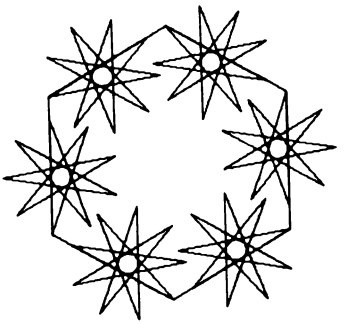
```
REPEAT 3 [CSILLAG 40 BAL.IV 180 (-10) CSILLAG 40
JOBB.IV 180 (-10)].
```



Figyeljük meg, mekkora hatást lehet elérni az elforgatás beiktatásával.

```
REPEAT 6 [CSILLAG 40 BAL.IV 180 (-10) RIGHT 30
CSILLAG 40 JOBB.IV 180 (-10) RIGHT 30].
```

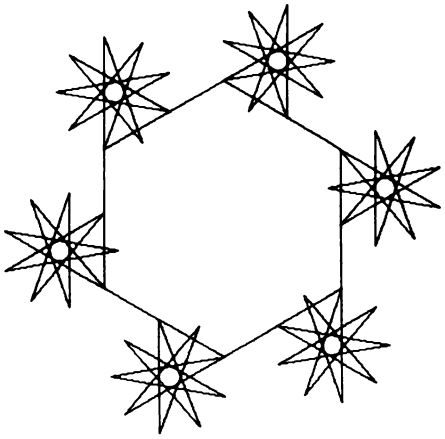




Lebilincselő – reméljük az Olvasónak is –, hogy egy-egy sorral most már micsoda útvonalakra tudjuk a Teknőcöt kényszeríteni. Beépíthetjük a csillagot pl. egy egyszerű hatszögbe is.

```
REPEAT 6 [FORWARD 60 RIGHT 60 CSILLAG 50].
```

REPEAT 6 [RIGHT 60 FORWARD 60 CSILLAG 50].



Figyeljük meg, hogy csupán az eltolás és forgatás felcserélésével, hogyan módosult az ábra!

A további látványos kísérleteket az Olvasóra bízva rátérünk a Teknőc legnevezetesebb ábráinak bemutatására. Újságcikkek, tanulmányok, ezeket szokták kiemelni a Logo, ill. a Teknőc-grafika hatékonyságának demonstrálására.

Első lépésben ismét egy poligont meghatározó eljárást adunk meg, amellyel egyaránt lehet konvex és konkáv szabályos sokszögeket előállítani.

Az eljárás abban különbözik az eddigiektől, hogy rekurzívan határozza meg a görbét.

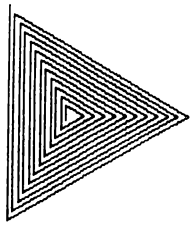
```
TO POLY :MERET :SZOG
 FORWARD :MERET
 RIGHT :SZOG
 POLY :MERET :SZOG
END
```

Ezt most már csak ki kell egészítenünk úgy, hogy kényelmes módosításokkal lehessen előállítani az ábrák sorát. Változtassuk folyamatosan a méretet!

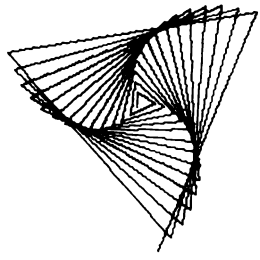
```
TO POLYSPI :MERET :SZOG :MVALTOZAS
 FORWARD :MERET
 RIGHT :SZOG
 POLYSPI ( :MERET + :MVALTOZAS ) :SZOG :MVALTOZAS
END
```

Lássunk néhány próbát!

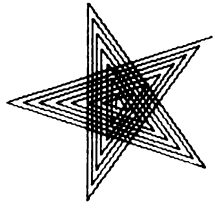
POLYSPI 5 120 2



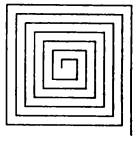
POLYSPI 5 122 2



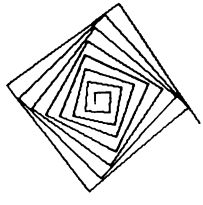
POLYSPI 5 144 2



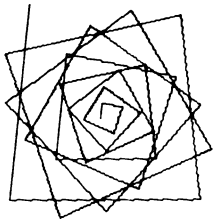
POLYSPI 5 90 2



POLYSPI 5 92 2



POLYSPI 5 96 2



Az INSPI eljárás a POLYSPI párja, csupán a méretváltozás helyett a szögváltozást adjuk meg.

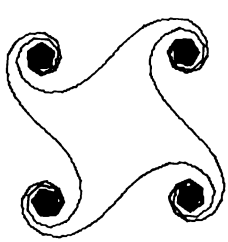
```
TO INSPI :MERET :SZOG :SZVALTOZAS
 FORWARD :MERET
 RIGHT :SZOG
 INSPI :MERET :SZOG + :SZVALTOZAS :SZVALTOZAS
END
```

Igen meglepő eredményeket kapunk különböző bemeneti értékre. Íme néhány ízelítőül

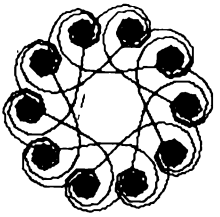
INSPI 5 10 10

INSPI 2 45 2

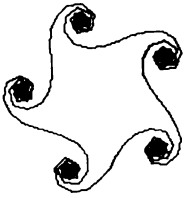
INSPI 5 45 4



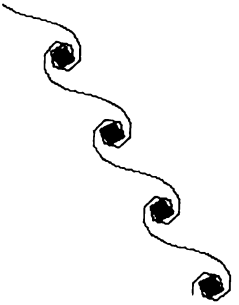
INSPI 5 44 5



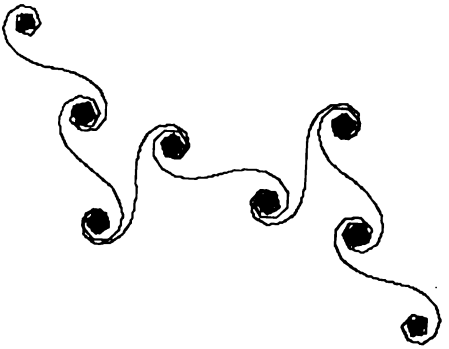
INSPI 5 33 10



INSPI 5 45 10



INSPI 5 10 7



Hogyan magyarázható a Teknőc viselkedése? Elemezzünk végig egy tevékenységsorozatot pl. az INSPI 5 10 10 eljárásban.

RIGHT 10

FORWARD 5

RIGHT 20

FORWARD 5

.

.

.

RIGHT 170

FORWARD 5

RIGHT 180 itt fordul meg a Teknőc

FORWARD 5

RIGHT 190 = LEFT 170, ami ugyanarra a szakaszra tereli, mint amit a fordulás előtt tett meg.

FORWARD 5

- Mitől függ a gombócok száma az ábrán?
- Van-e olyan paraméterhármas, amely mellett a Teknőc sosem tér vissza a már megrajzolt görbére?

A kérdések megválaszolásához figyelembe kell vegyük, hogy az INSPI eljárás a POLY leszármazottja. Így a gombócok a POLY csúcsainak felelnek meg. Tekintsünk tehát el a cirkalmaktól, és vizsgáljuk meg csúcstól-csúcsig a Teknőc kezdő és végállapotát!

Vegyük példának az INSPI 5 45 4 ábráját, és kövessük végig a szögnövekedés lépéseit!

RIGHT 45 kezdőszög

RIGHT 45 + 4

RIGHT 45 + 2 * 4

.

RIGHT 45 + 89 * 4

RIGHT 45 + 90 * 4 = 360, amelynek hatása azonos az elsővel. Tehát 90 lépés után a ciklus megismétlődik.

A ciklus összfordulatszámát kiszámolva

$45 + (45 + 4) + (45 + 2 * 4) + \ldots (45 + 89 * 4) = 12060.$

Ezt 360°-kal elosztva a maradék 270°. Vagyis egy ciklus összfordulatszáma 270°, ill. −90°. Így már választ kapunk arra, hogy miért négyszögű az INSPI 5 45 4 ábrája.
Általánosan megvizsgálva az

INSPI : MERET : SZOG : SZVALTOZAS

eljárást, ha a SZVALTOZAS = 360°/n, ahol n egész szám az egyes ciklusok utáni összfordulat.

$n * SZOG + SZVALTOZAS * n * (n-1)/2 = n * SZOG + (n-1) * 180°$

alapján számolható.

Ez a tulajdonság konvex és konkáv INSPI-görbékre egyaránt igaz.

Vizsgáljuk meg most azt az esetet, amikor az összfordulat egy ciklus után nulla, mint pl. az INSPI 2 45 2 esetében. Az ábra ekkor a POLY : MERET O összképét mutatja, vagyis egy egyenes mentén jönnek létre a gombócok, tehát a görbe sosem záródik.

A POLYPSI végtelen eljárását némi módosítással végessé tehetjük, és ilyen módon újabb típusú görbéket is kapunk: spirálisok spiráljait, amelyeket spirolaterálisoknak hívnak.

```
TO POLYSPIRO :LEPES :MAX :MERET :SZOG :NOV
 IF :LEPES > :MAX THEN STOP
 FORWARD :MERET
 RIGHT :SZOG
 POLYSPIRO ( :LEPES + 1 ) :MAX ( :MERET + :NOV ) :SZOG :NOV
END

TO SPIRO :MAX :MERET :SZOG :NOV
 POLYSPIRO 1 :MAX :MERET :SZOG :NOV
 SPIRO :MAX :MERET :SZOG :NOV
END
```

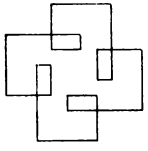
A POLYSPIRO a POLYPSI első MAX vonalait rajzolja meg, amelynek befejeztével az összfordulat:

MAX * SZOG

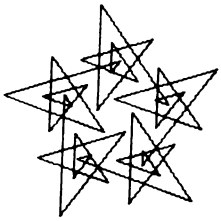
A SPIRO ismétli meg a POLYSPIRO ciklusát.

Ezekkel az eljárásokkal is olyan eszközt adunk az Olvasó kezébe, amellyel érdekes kísérletet hajthat végre a bemutatott rajzokon túl.

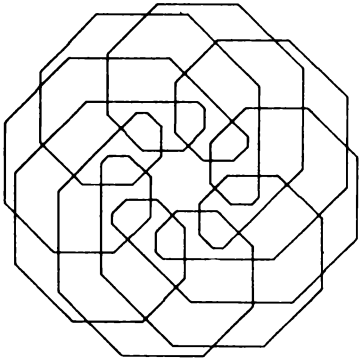
SPIRO 5 5 90 5



SPIRO 8 5 144 5



SPIRO 11 5 45 5



1.5.4. Rekurzív görbék

Befejezésül a rekurzív görbékre adunk eljárást.

Vegyünk pl. egy adott hosszúságú szakaszt és osszuk pl. 3 egyenlő részre.

Vegyünk egy mintát: a középső szakaszt 60°-os szögben megtörjük.

```
TO G1 :MERET
 FORWARD :MERET / 3
 LEFT 60
 FORWARD :MERET / 3
 RIGHT 120
 FORWARD :MERET / 3
 LEFT 60
 FORWARD :MERET / 3
END
```

Az így kapott töröttvonal, minden szakaszát továbbosztjuk 3 egyenlő részre és a középsőket 60°-os szögben ismét megtörjük.

```
TO G2 :MERET
 G1 :MERET / 3
 LEFT 60
 G1 :MERET / 3
 RIGHT 120
 G1 :MERET / 3
 LEFT 60
 G1 :MERET / 3
END
```

A töröttvonal szakaszait ismét osszuk 3 egyenlő ... és így tovább.

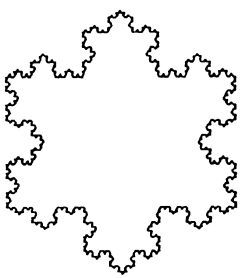
```
TO G3 :MERET
 G2 :MERET / 3
 LEFT 60
 G2 :MERET / 3
 RIGHT 120
 G2 :MERET / 3
 LEFT 60
 G2 :MERET / 3
END
```

Az eljárás ugyan akármeddig folytatható, de a rajzolhatóság miatt általában csak néhány lépésig terjed (hiszen exponenciálisan csökken a szakasz hossza).

A teljesen azonos felépítésű eljárásokat egy eljárással helyettesítjük, amelynek a bemeneti paramétere a szakasz hosszán kívül a rekurzió szintje is.

```
TO GORBE1 :MERET :SZINT
 IF :SZINT = 0 THEN FORWARD :MERET STOP
 GORBE1 :MERET / 3 :SZINT - 1
 LEFT 60
 GORBE1 :MERET / 3 :SZINT - 1
 RIGHT 120
 GORBE1 :MERET / 3 :SZINT - 1
 LEFT 60
 GORBE1 :MERET / 3 :SZINT - 1
END
```

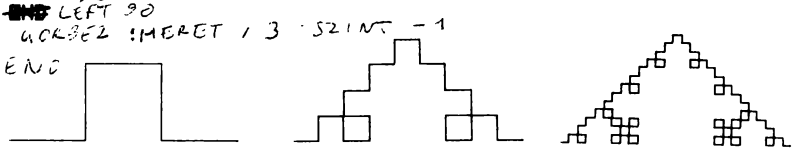
Előállíthatunk így egy rekurzív görbével határolt alakzatot is.



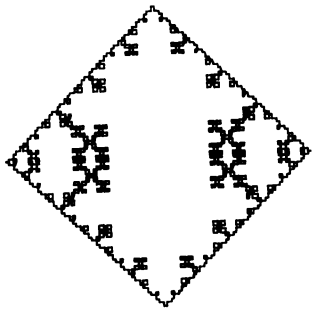
```
TO GORBE1.HAROMSZOG :MERET
 REPEAT 3 [GORBE1 :MERET 4 RIGHT 120]
END
```

Végül minden kommentár nélkül mutatunk be néhány rekurzív görbét a hozzá tartozó eljárással.

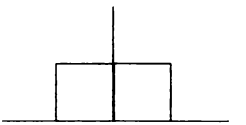
```
TO GORBE1 :MERET :SZINT
 IF :SZINT = 0 THEN FORWARD :MERET STOP
 GORBE1 :MERET / 3 :SZINT - 1
 LEFT 60
 GORBE1 :MERET / 3 :SZINT - 1
 RIGHT 120
 GORBE1 :MERET / 3 :SZINT - 1
 LEFT 60
 GORBE1 :MERET / 3 :SZINT - 1
END
```

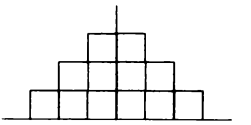


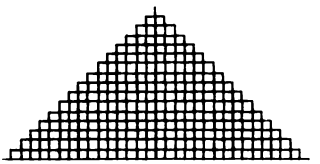
```
TO GORBE1.HAROMSZOG :MERET
 REPEAT 3 [GORBE1 :MERET 4 RIGHT 120]
END
```

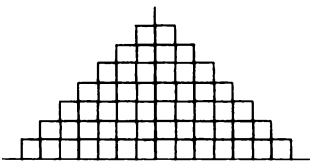


```
TO GORBE3 :MERET :SZINT
 IF :SZINT = 0 THEN FORWARD :MERET STOP
 GORBE3 :MERET / 2 :SZINT - 1
 LEFT 90
 GORBE3 :MERET / 2 :SZINT - 1
 RIGHT 180
 GORBE3 :MERET / 2 :SZINT - 1
 LEFT 90
 GORBE3 :MERET / 2 :SZINT - 1
END
```

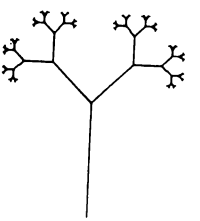








```
TO FA :MERET :SZINT
 IF :SZINT = 0 THEN STOP
 FORWARD :MERET
 LEFT 45
 FA :MERET / 2 :SZINT - 1
 RIGHT 90
 FA :MERET / 2 :SZINT - 1
 LEFT 45
 BACK :MERET
END
```



II. RÉSZ

2. Üdvözöljük a Logoban

3. Logo haladóknak

4. Példák nagyobb programokra

2. Üdvözöljük a Logoban

Kétségtelen, hogy a Logo leginkább a Teknőc-grafikáról ismert, kövessük mégis végig a hagyományos számítógépes nyelvtanulás útját. Először a nyomtatási utasításokkal foglalkozunk, majd ismét érintve a grafikát, a műveletek definiálási módjai után, a Logo sajátos vizsgálószavait ismertetjük. Még ebben a fejezetben tárgyaljuk a feltételes, az ugróutasításokat is.

2.1. Közvetlen utasítások

2.1.1. A számítógép mint zsebszámológép

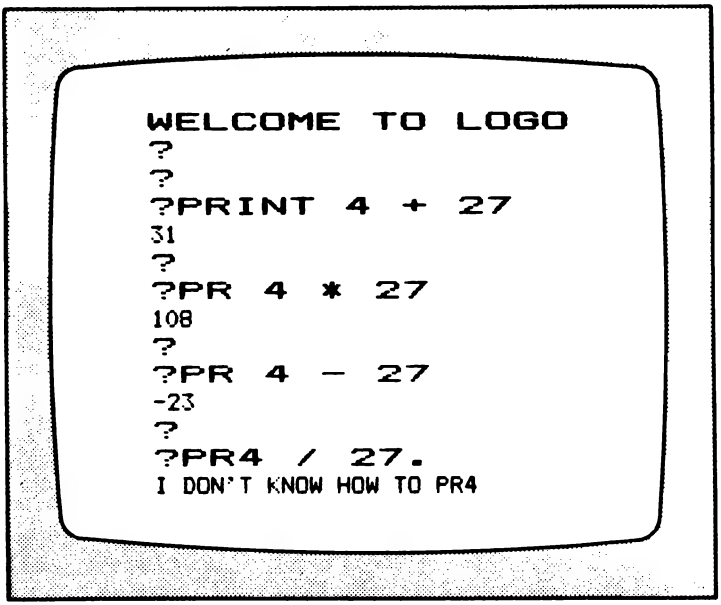
Nyilvánvaló, hogy minden személyi számítógép zsebszámológépként is használható, csak alapvető működésmódját kell hozzá ismernünk. Ezt az alapvető műveletek segítségével mutatjuk be.

[4] [X] [2] [7] [=]

Amennyiben megnyomjuk egy zsebszámológép ezen öt gombját, akkor az [=] nyomógomb lenyomása után megjelenik az eredmény: 108.

A személyi számítógépen rendszerint az [=] nyomógomb helyett a [RETURN] nyomógombot kell megnyomni. Ez a nyomógomb jelzi a számítógépnek, hogy a bevitt sornak vége van, és azt végre kell hajtani. Ezenkívül a Logoban kifejezetten meg kell mondanunk, hogy az eredményt ki is kell nyomtatni. A Logo kiszámítja ugyan, de az eredménynek sehol sincs nyoma! További magyarázgatás helyett próbáljunk ki most kisebb számítási feladatokat. Számoljuk és nyomtassuk ki a következőket:

4 + 27
4 X 27
4 – 27
4 : 27 .



Az újra és újra megjelenő kérdőjel azt jelzi, hogy adatokat vihetünk be. A kérdőjelet válaszjelnek is nevezzük. Tehát ha megjelenik a válaszjel, akkor tudjuk, hogy a Logo befejezett egy feladatot és további adatra vár.

A képernyőn levő villogó jel (kurzor) azt a helyet jelöli, ahol a következő begépelendő karakterünk meg fog jelenni. A kurzor célja, hogy megkönnyítse a képernyőn a munkát.

A képernyőn látható apró betűs szöveg a számítógép válasza a bemenetre.

Már itt az első példák között is különbségeket figyelhetünk meg, láthatjuk, hogy a személyi számítógépünk az utolsó példában valamit nem ért, rögtön *visszabeszél.* Ezek és sok más üzenet hívja fel a figyelmünket az apró kezelési, később pedig programozási hibáinkra. Csak gondosan el kell olvasnunk az üzeneteket, és máris tudjuk, hogy mit jelentenek.

Mi történt itt? A korábbi sorokkal ellentétben a PR és a 4 között nem hagytunk üres helyet (amit blank-nek is neveznek) a szóköz billentyű meg-

nyomásával. A szóközök a Logo számára létfontosságúak. A szóköz alapján ismeri fel ui. a Logo, hogy egy utasítás vagy egy szám begépelése befejeződött. A Logo csak a PRINT nyomtatási utasítást vagy ennek rövid alakját, a PR-t ismeri. PR4 teljesen ismeretlen számára. Így a Logo nem képes a PR4 utasítást végrehajtani. („Nem tudom, hogyan kell PR4-ni" – ezt jelenti a kinyomtatott hibaüzenet.)

A Logoban az alapműveletek műveleti jelei mások. Tizedes számok esetében a zsebszámológéphez hasonlóan pontot kell használni, különben máris megjelenik a hibaüzenet ...

A hibánkat kijavítottuk, beiktattuk a szóközt. Az osztás eredménye tizedes szám.

```
?
?
?
?
?
?PR 4/27
.148148
?PR 4=27
FALSE
?PR 4<27
TRUE
?PR 4>27
FALSE
?-4*(4-27)
I DON'T KNOW WHAT TO DO WITH 92
?
```

Az egyenlőségjel a Logo számára mást jelent, mint a zsebszámológépen. A < (kisebb mint) és a > (nagyobb mint) jelhez hasonlóan relációs operátor.

Ilyen összehasonlítások után két lehetséges eredmény közül az egyik jelenik meg. Az összehasonlítás eredménye vagy TRUE (igaz), vagy FALSE (hamis).

A befejezésnél a hibaüzenet azért jelent meg, mert megfeledkeztünk a nyomtatási utasításról. A Logo elvégezve a számítási műveleteket, jelenti: „Nem tudom, mit tegyek a 92-vel".

2.1.2. Amit a zsebszámológép már nem tud

Már az előző szakaszban szereplő szám-összehasonlítások is olyan lehetőségek, amelyekkel egy egyszerű zsebszámológép nem rendelkezik. A Logo azonban egyes jeleket, szavakat és mondatokat is ki tud nyomtatni.

```
?PR "4
4
?PR "COMPUTER
COMPUTER
?PR [A B C]
A B C
?PR []

?PR [[A B] C []]
[A B] C []
?PR "SZOö öÄö<ö=
SZO Ä<=
```

Egy magában álló szó mindig az idézőjelről ismerhető fel. Az ilyen szavak egy vagy több jelből állnak. A számok ugyancsak lehetnek így szavak. Tehát idézőjellel és anélkül is bevihetők.*

Több szó mondatot képez. A mondat mindig szögletes zárójelben van. A mondat egy vagy több szóból tevődik össze. A mondat további részmondatokból állhat. Az ilyen mondatot a Logoban listának nevezik. A szavak és a mondatok speciális esete, ha azok nem tartalmaznak betűket, ill. szavakat vagy részmondatokat, vagyis ha üresek.

Gépeljünk be egy szót idézőjel nélkül. Ekkor válaszul a már ismert hibaüzenetet kapjuk. Ha tehát egy szót idézőjel nélkül írunk be, akkor a Logo ezt a szót először utasításnak (mint a PRINT vagy PR) tekinti, de kénytelen megállapítani, hogy ilyen utasítást vagy feladatot nem ismer, tehát nem tudja, hogy ...

A mondatok (listák) kinyomtatásakor a két külső zárójelet a gép nem nyomtatja. Az [[A B] C []] mondat (lista) három részből áll: az [A B] listából, a "C szóból és a [] üres listából. A mondatokon belül a szavak idézőjeleit nem kell kirakni. A szavak idézőjel nélkül nyomtatódnak ki.

Figyeljük meg az utolsó két különös sort. Egy olyan szót tartalmaz, amelyik a speciális jeleket is tartalmazza. Amennyiben a szavakban ilyenek előfordulnak, akkor a Logoval közölnünk kell, hogy kivételesen az adott speciális jelet közönséges jelnek kell tekinteni. Ezen egyedi jelek – éppúgy, mint a szóközök – a Logo számára elválasztóként hatnak. Az elválasztó tulajdonság a CTRL-Q beírásával megszüntethető. (A CTRL nyomógombot és a Q nyomógombot egyidejűleg meg kell nyomni, azaz a CTRL-t lenyomva kell tartani, majd Q-t kell ütni.) A Logo minden CTRL-Q-t ö betűvel jelölt. Az eredménysorban természetesen az ö már nem jelent meg.

* A [zárójel helyett Ä a] zárójel helyett Ü van itt, és a további példákban is egészen a 185. oldalig (Lektor).

2.1.3. Ismerkedés a műveletekkel – Szavak és listák feldolgozása

A következő példák szavakon és listákon végrehajtható négy műveletet mutatnak be, a műveletek eredményét nyomtatási utasítással íratjuk ki. Vizsgáljuk meg közelebbről ezeket a példákat.

```
?
?
?
?
?PR FIRST "LOGO
L
?PR FIRST [A B] C
A B
?PR BUTFIRST "LOGO
OGO
?PR BF [A B C]
B C
?PR BF [[A B] C]
C
?PR COUNT [A B C]
3
?PR EMPTYP []
TRUE
?
```

A FIRST művelet minden esetben a bemenet első elemét adja: a szó első betűjét vagy a lista első elemét. A BUTFIRST művelet (rövid alak: BF) visszaadja a bemenetet, az első elemet kivéve. A COUNT művelet megszámolja a lista elemeit, és eredményül egy számot ad. Az EMPTYP művelet megvizsgálja a bemenetet, hogy üres lista vagy üres szó-e, eredményül pedig TRUE (igaz) vagy FALSE (hamis) adódik.

Írjon most be egy egyelemű listát és egy egyetlen jelből álló szót! Figyelje meg a FIRST, BF és COUNT eredményét. A FIRST eredményül a betűt vagy a lista tartalmát adja. A BUTFIRST üres szót vagy üres mennyiséget ad. Egyelemű lista esetén a COUNT eredménye 1. Amennyiben azonban a COUNT bemeneteként egy szót adunk meg, akkor a következő hibaüzenet jelenik meg: „COUNT DOESN'T LIKE...AS INPUT". A hibaüzenet arra utal, hogy a COUNT művelet esetében a ... bemenet nem megengedett.

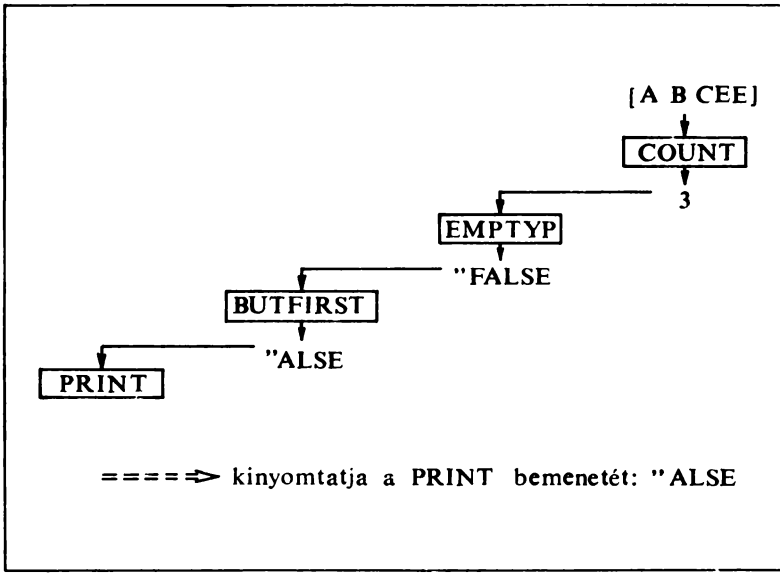
Vizsgáljuk meg azt is, hogy miképpen viselkedik e négy művelet üres szavak és listák esetén. A gép minden esetben *visszabeszél,* ami azt jelenti, hogy ezek a bemenetek nem megengedettek. A Logo sem varázsló, hogy az üres lista első eleméül valami határozott dolgot adjon. A COUNT művelet üres lista esetén így 0-t ad, ami azt jelenti, hogy ennek a listának egyetlen eleme sincs.

Eddig a művelet eredményét azonnal kinyomtattuk. Azonban, mint ahogy már említettük, valamely művelet eredménye maga is egy további művelet be-

menete lehet stb. Tehát – mindaddig, amíg van értelme – tetszőleges számú műveletet láncolhatunk.

A műveletek egymás mellé sorolása eleinte zavarosnak tűnik. Egyszer azonban gondosan át kell rágnunk magunkat rajta, és tudomásul kell vennünk, hogy a műveletek bemenete más műveletek eredménye lehet.

Eleinte még egyszerű a dolog. Minden egyes műveletet egy doboznak tekintünk. Minden egyes doboznak egy (vagy több) bemenete van, kimenetként pedig megadja az eredményt. Az eredmény a következő doboz bemenete.



Az ábrázolás alapján úgy vélhetné az ember, hogy egy ilyen, láncolt műveletekből álló Logo sort jobbról balra kell olvasni. Ennek ellenére maradjunk régi szokásunknál, olvassuk a sorokat balról jobbra. Példánkban ez azt jelenti, hogy minden, ami a PRINT utasítástól jobbra áll, az a bemenete. A PRINT utasítás bemenete a valamin végrehajtott BUTFIRST művelet. És ez a valami az EMPTYP művelet, amelynek bemenete COUNT [A B [C]] .

Vizsgáljuk meg a következő példákat:

PRINT BF BF [A B [C]] ;

PR FIRST BF BF [A B [C]] ;

PR FIRST FIRST BF BF [A B [C]] .

Mindhárom Logo sor eredménye egy C. Az első sorban a háromelemű listából az első két elemet eltávolítjuk. Visszamarad egy egyelemű lista, mégpedig [[C]].

A második Logo sorban a FIRST meghatározza a lista első elemét, vagyis [C]-t.

A harmadik sorban a további FIRST megkeresi a beadott lista első elemét, vagyis a „C" szót. Ne feledjük, hogy nyomtatáskor nem jelenik meg a listák két külső zárójele, ill. a szavak idézőjele.

2.1.4. Hibajavítás

Biztos vagyok benne, hogy mindannyian követtünk már el gépelési hibát. A RETURN nyomógomb lenyomása után ilyenkor egy hibaüzenet jelent meg. A kívánt hatást természetesen nem értük el. Tehát az egész sort újra be kellett gépelnünk. Létezik azonban számos olyan vezérlő nyomógomb, amely segíthet.

Ha elfelejtettük beírni a szóközt, a sort nem kell újra leírnunk, még ha megnyomtuk is a RETURN gombot. Nyomjuk meg CTRL-Y-t (a CTRL gombot lenyomjuk, közben megnyomjuk az Y gombot). Ekkor megjelenik a hibás bemeneti sor. Nyomjuk meg a RETURN-t. És mégegyszer a CTRL-Y-t. Ekkor az utolsó bemeneti sor ismét módosítható.

Milyen lehetőségek vannak még a kurzor mozgatására?

CTRL-B-vel a sorban jelenként visszalépegethetünk a sor elejéig.

A → nyomógombbal pedig jelenként jobbra haladunk, ha a sorban még vannak jelek. (Próbáljunk meg most CTRL-B-vel visszalépegetni a sor elejére.)

CTRL-E-vel közvetlenül a sor végére ugorhatunk, és így megtakaríthatjuk a → nyomógomb többszöri megnyomását, ha a bementi sort rögtön RETURN-nel akarjuk lezárni.

CTRL-A-val pedig közvetlenül a sor elejére ugrunk.

Most már a kurzort a bemeneti sor tetszőleges helyére vihetjük. De hogyan illeszthető be egy jel? Egyszerűen lépegessünk vissza a kurzorral oda, ami elé a szóköz szükséges és nyomjuk meg a szóköz billentyűt.

Legjobb, ha feljegyezzük magunknak ezeket a nyomógombokat, mert hozzásegítenek a gépelési hibák kényelmes javításához.

2.1.5. A TYPE és a PRINT utasítás

A Logoban egy bemeneti sorban több utasítás állhat egymás mellett. Próbáljuk ki a TYPE és a PRINT hatását

```
PR "HEL PR "YES , ill.
TYPE "HEL TYPE" YES
```

beírásával.

A PRINT utasításból álló sor azt mutatja, hogy a nyomtatási művelet végrehajtása után a nyomtató a következő sor elejére áll. A képernyőn ennek megfelelően a kurzor egy sorral lejjebb a sor elejére áll. Ez jól látható a kérdőjelen is, amely a "YES kinyomtatása után egy betűvel lejjebb, bal oldalon jelenik meg. A TYPE utasítást artalmazó példa mutatja, hogy a nyomtatási művelet után a kurzor vagy a nyomtatófej az utolsó kiírt betű mellett megáll.

A gömbölyű zárójelbe téve az egész bementi sort, elég csak egyszer beírni a PRINT ill. TYPE utasítást. A PRINT a kiíratás során a bemenetek közé soremelést tesz. A TYPE ellenben a kiírandó szavakat szóköz nélkül írja.

2.1.6. Összefoglalás és gyakorlatok

Az első fejezetben a Logo nyelvtan nagy részét megtárgyaltuk, és a Logo szókincséből már jó párat ismerünk. Foglaljuk össze a legfontosabbakat!

1. A Logoban utasítások és műveletek vannak, amelyek bemenete szó, ill. lista lehet.
2. A szó jelölésére idézőjel szolgál, a szó egy vagy több jelből áll. A szó tehát számok, betűk és egyedi jelek sorozata. Különleges eset az üres szó A számok ugyancsak szavak, de ezeket idézőjel nélkül is be lehet írni.
3. Szögletes zárójele közé (Ä és ü a nyomtatási képe esetünkben) a listát (mondatot) írjuk. A lista elemei (részmondatok) szavak, ill. további listák lehetnek. Különleges eset az üres lista.
4. A szavak és a listák kinyomtatásakor az idézőjelek és a lista két külső zárójele nem jelenik meg. A listában a szavak idézőjel nélkül állnak.
5. A Logo teljes utasítássora egy utasítással (pl. PR, TYPE ...) és a bemeneteivel (szavak, listák, műveletek) kezdődik. A bemeneti sor a RETURN nyomógombbal zárható le.
6. A szóköz elválasztójel. Szóközök választják el egymástól a bemeneti sorban az utasításokat, műveleteket, szavakat és listákat. Ha megfeledkezünk a szóközökről, akkor a Logo az egyes részeket nem tudja megkülönböztetni. Elválasztójelként szerepel még néhány különleges jel (a zárójelek, a műveleti jelek, a relációs jelek stb.). Amennyiben a sza-

vakban szóközöknek vagy más elválasztó tulajdonsággal rendelkező jeleknek kell előfordulniuk, akkor a bemenet során előzőleg CTRL-Q-t kell beírni (ö betű a nyomtatási képe!). Ez megszünteti a következő jel elválasztó tulajdonságát.

7. Egy bemeneti sorban egymás mellett több nyomtatási utasítás állhat.
8. A PRINT és a TYPE nyomtatási utasítások egyetlen vagy tetszőleges számú bemenettel rendelkezhetnek; az utóbbi esetben gömbölyű zárójelek közé kell írni őket.
9. A műveletek láncolhatók, azaz valamely művelet bemeneteként más műveletek által adott eredmény állhat.
10. A műveletekből nem lehet kifelejteni a szóközt vagy az idézőjelet. A nyomtatási utasítások hatására ilyenkor leggyakrabban a következő hibaüzenetek jelennek meg:

    "I DONT'T KNOW HOW TO..."
    "I DONT KNOW WHAT TO DO WITH..."
    "...DOESN'T LIKE... AS INPUT" .

11. A vezérlőgombokkal a bemeneti sor tetszőleges betűjére léphetünk. A jelek törölhetők és újak írhatók be.

Új Logo szavak:

PRINT	COUNT
PR	EMTYP
+	TYPE
–	(PRINT)
+	(TYPE)
/	
>	CTRL-A
<	CTRL-B
=	CTRL-E
CTRL-Q	CTRL-Y
FIRST	→ nyomógomb
BUTFIRST	← nyomógomb
BF	

Feladatok

1. Írjon be egy négyelemű listát, amely két szóból, egy üres és egy három szóból álló listából áll!
2. Ellenőrizze az 1. feladat megoldását úgy, hogy a COUNT műveletet írja elé!

3. Írassa ki a COMPUTER szó ötödik jelét!
4. Írjon be egy olyan szót, amely 10 egyedi jelből áll (szóközök, gömbölyű és szögletes zárójelek, műveleti jelek)!
5. A függelékből keressen ki három további, szavak és listák kezelésére alkalmas műveletet és tesztelje őket!
6. Szándékosan hibás vagy nem megengedett bemenetek segítségével idézze elő az eddig ismert három hibaüzenetet!
7. Írjon be egy olyan sort, amely több PRINT és TYPE utasításból áll! A RETURN gomb megnyomása előtt mondja meg, hogy mi lesz az eredmény!
8. Mi a válasz a PRINT (COUNT [[1 2 3] [] [ÉN TE]])* (COUNT [A B C]) utasításra?

2.2. Egyszerű nyomtató programok

2.2.1. Egy versike

Az első fejezetben megismerkedtünk a Logo szavaival és ezek utasításokká való összeállításával. Minden alkalommal, amikor a RETURN nyomógombbal egy bemeneti sort lezárunk, a sor elvész, azaz ha újra le akarjuk futtatni, akkor újra be kell gépelnünk. A két új szó, a TO és az END segítségével tetszőleges számú utasítássorból állíthatunk össze programot. Ezt a programot bármikor előhívhatjuk és futtathatjuk.

Kezdjük egy kis nyomtatóprogrammal. A program PRINT utasításokból áll. Ha a programot le akarjuk futtatni, akkor csak a program nevét kell leírnunk, és meg kell nyomnunk a RETURN nyomógombot.

A Logo ezeket a felhasználó által definiált funkciókat (programokat) éppúgy kezeli, mint a Logo rendszerfunkcióit (pl. PRINT, TYPE, FIRST stb.). A funkciók hívásakor egyszerűen beírjuk a nevét, idézőjelre vagy más ilyesmire nincs szükség.

Hogyan definiálhatunk tehát saját programokat? Először a TO segítségével a számítógépnek jeleznünk kell, hogy most részletesen elmagyarázunk neki egy programot, mindaddig, amíg be nem írjuk az END-et. A TO, a programnév és a RETURN beírása után megváltozik a válaszjel. ? helyett most > jelenik meg. A Logo rendszer tanulásra kész, ún. editáló üzemmódban van. Most soronként beírjuk azt, amit szeretnénk.

A definíciós műveletet END-del kell lezárni. A Logo ilyenkor nyugtázza, hogy elmagyaráztuk neki a VERS programot.

```
?TO VERS
>PR "DUNA VIZE OLAJOS "
>PR "FURODJ BENNE ALAJOS "
>END
VERS DEFINED
?
?
?
?
?VERS
DUNA VIZE OLAJOS
FURODJ BENNE ALAJOS
?
```

Amennyiben nem követtünk el gépelési hibát, első kis nyomtató programunk (a név beírása és a RETURN gomb lenyomása) után lefut.

Aki „elütötte magát", az kénytelen áttanulmányozni a 2.2.5. pontot, hogy a programban levő hibákat utólag kijavíthassa.

2.2.2. Címzés

Most már tetszés szerinti nyomtató programokat is készíthetünk. Definiáljunk egy olyan funkciót, amely kinyomtatja a címünket! A nyomtatási művelet elején és végén írassunk ki egy-egy üres sort!

```
?
?TO CIM
>PR 
>PR "KOVACS JANOS "
>PR "KOSSUTH UTCA 1 "
>PR "VESZPREM 8200 "
>PR "
>END
CIM DEFINED
?
?
?
?CIM
```

Ellenőrzésként írjuk be rögtön ezt az utasítássort is, és figyeljük meg, hogy mi történik!

```
KOVACS JANOS
KOSSUTH UTCA 1
VESZPREM 8200

?
```

2.2.3. Geometriai ábrák és nyomtatott képek

Éppúgy, mint a két előző programban, természetesen soronként definiálhatunk háromszögeket, vagy bármilyen más ábrát.

A programban a szóközt is használjuk, de hogy ne hasson elválasztójelként, minden esetben a CTRL-Q segítségével az elválasztó tulajdonságot fel kell függesztenünk.

```
?
?TO PIRAMIS
>PR "ö ö ö 3
>PR "ö ö 333
>PR "ö 33333
>PR "3333333
>END
PIRAMIS DEFINED
?
?
?
?
?
?
?
?PIRAMIS
   3
  333
 33333
3333333
?
```

2.2.4. Minden megismételhető

Például tetszőleges számú öntapadó címkét akarunk kinyomtatni. Erre a REPEAT utasítás igen alkalmas. Működéséhez két bemenet szükséges: az első az ismétlések száma, a második egy lista különböző utasításokkal. A szám és a lista közé természetesen szóközt kell tenni.

Írassunk ki három címkét Kovács Jánosnak!

```
?REPEAT 3 [ CIM ]

KOVACS JANOS
KOSSUTH UTCA 1
VESZPREM 8200

KOVACS JANOS
KOSSUTH UTCA 1
VESZPREM 8200

KOVACS JANOS
KOSSUTH UTCA 1
VESZPREM 8200
```

Ábrák és képek tetszőleges keverékét is akárhányszor kiírathatjuk.

2.2.5. Programok módosítása a Logo editorral

Az 1. fejezetben már megismerkedtünk egy csomó javító nyomógombbal. Ezek azonban csak egy, RETURN-nel még nem lezárt bemeneti sorban használhatók. Hogyan férünk hozzá a program egy már definiált hibás sorához? Erre a célra az EDIT utasítás (rövidítve: ED) szolgál.

Az EDIT utasítás után idézőjellel be kell adni a kívánt programnevet, majd a RETURN-t.

```
?
?
?ED "VERS
VERS DEFINED
?
?
?PO "VERS
TO VERS
PR [DUNA VIZE OLAJOS]
PR [FURODJ BENNE ALLAJOS]
END

?
?
?
?ED "VERS
VERS DEFINED
?PO "VERS
TO VERS
PR [DUNA VIZE OLAJOS]
PR [FURODJ BENNE ALAJOS]
END
```

A Logo jelzi, hogy valóban megvan a program, amit ekkor a PO" programnév utasítással listázhatunk ki (PO = Print Out).

Ekkor a már ismert javító gombokkal módosíthatunk vagy új szöveget iktathatunk be. Elvileg a két vezérlő nyomógomb is megteszi. A → nyomógombot folyamatosan megnyomva tartva bal oldalon felülről az END-ig végigszaladhatunk az egész programon. Ugyanez fordítva a CTRL-B-vel végezhető el. Ha helyette a ← nyomógombot használtuk volna, akkor kitöröltük volna az egész szöveget. Javítsuk most ki a hibákat! Menjünk a kurzorral a hibás helyekre és végezzük el a módosításokat! A Logo editorból a CTRL-C-vel léphetünk ki.

A Logo editor használatakor a CTRL-N-nel a kurzor a következő sor azonos pozíciójára léptethető.* Soronként a CTRL-P-vel léphetünk vissza. CTRL-E-vel a mindenkori sor végére lépünk. A további editáló utasításokat a kézikönyvekben találhatjuk meg.

* Teszteléskor ez nem működött (Lektor).

2.2.6. Összefoglalás és gyakorlatok

1. A Logoban a felhasználó által definiált funkciók (programok) tetszőleges, Logo szóként szereplő nevet kaphatnak.
2. A Logo rendszernek a funkciókat (programokat) meg lehet tanítani.
3. Ezt a TO és a programnév beírása váltja ki.
4. Ilyenkor a válaszjel ?-ről >-re változik.
5. A felhasználó által definiált funkció tetszőleges számú utasítássorból állhat. Minden egyes utasítássort RETURN-nel, a műveletet END-del zárjuk. Ekkor ismét megjelenik az eredeti válaszjel.
6. A definiálást a Logo nyugtázza.
7. A felhasználói funkció végrehajtatása úgy történik, hogy beírjuk a felhasználói funkció nevét (idézőjel nélkül, éppúgy, mint egy eredeti Logo funkció esetében).

Új Logo szavak:

TO
END
REPEAT
PO
EDIT
ED
CTRL-C
CTRL-N
CTRL-P

Feladatok

1. Készítsen olyan nyomtató programot, amely tetszőleges szöveget (egy versszakot, egy éneket vagy egy megjegyzést) ír ki!
2. Egy címprogram segítségével írassa ki a számítógéppel a címét és a telefonszámát!
3. Készítsen négyszög alakú keretet X-ekből!
4. Nagyított alakban írassa ki, hogy OK!

2.3. Bővülő programok

2.3.1. Programok programokat hívnak

Egyetlen vezérlő programban összekapcsolhatjuk a külön-külön meglevő programjainkat. Ez lesz a módja a nagyobb lélegzetű programok megírásának is.

Egyszerű példaként a már meglevő programjainkat összekapcsolva, készíthetünk egy több versszakból álló verset, egy képkiállítást, egy névjegyzéket.

Hogyan hívhatók elő a már meglevő programok? A POTS (Print Out Procedures), ill. a POALL (Print Out All) utasítás a meglevő funkciók listáját adja meg.

```
?
?POTS
TO PIRAMIS
TO CIM
TO VERS
?
```

Az egyes programok listáját a PO "programnév utasítás segítségével hívhatjuk a képernyőre, amint azt már az editálásnál láttuk.

```
?
?PO "VERS
TO VERS
PR [DUNA VIZE OLAJOS]
PR [FURODJ BENNE ALAJOS]
END
```

Ez nyilván kényelmessé teszi a hasonló funkciók definiálását. Az egyes versszakokat, képeket, címeket természetesen külön funkciónévvel kell ellátni.

Egy vezérlő program első lépésében töröltetni kell a képernyőt: CLEAR SCREEN. Ezután következzék a funkciók felsorolása TO nélkül csak RETURN-nel elválasztva, majd a záró END előtt még egy üres lista: PR [], ami el fogja választani az egyes versszakokat, képeket, címeket.

Névjegyzék programunk most még csak egy nevet írat ki, és csupán egyetlen REPEAT utasításból áll. Nem akartuk azonban a programot végigfuttatni,

```
TO NEVJEGYZEK
REPEAT 10 [CIM]
END
```

ezért CTRL-G-vel megszakítottuk.

```
?NEVJEGYZEK

KOVACS JANOS
KOSSUTH UTCA 1
VESZPREM 8200
```

```
KOVACS JANOS
KOSSUTH UTCA 1
VESZPREM 8200

KOVACS JANOS
KOSSUTH STOPPED! IN CIM:
```

2.3.2. Programok bemenő adatokkal

Egy névjegyzékben amúgy is több névnek kell szerepelnie. Hogyan oldható ez meg anélkül, hogy hasonló programok tömegét kellene megírnunk?

A funkciónév megadásakor ugyanabba a bemeneti sorba felsoroljuk a funkcióbemenetek lehetséges megnevezéseit. Ezeket a paramétereket tartályoknak tekinthetjük, amelyeket csak később töltünk fel az aktuális tartalommal. A paraméter neve kettősponttal kezdődik. Programunk következő utasításaiban a paramétereket akárhányszor használhatjuk, de a paraméter nevek egyetlen jelben sem különbözhetnek a legfelső sorban megadott nevüktől!

```
TO CIM :NEV :IR.SZAM :HELYSEG :UTCA
PR "
PR :NEV
PR :HELYSEG
PR :UTCA
PR :IR.SZAM
PR "
END
CIM DEFINED

?CIM [ RETI VIRAG ] "6400 [ KISKOROS ] [ BEREK U 2 ]

RETI VIRAG
KISKOROS
BEREK U 2
6400
```

Néhány hibalehetőségre felhívjuk a figyelmet.

1. Tartsuk be a bemenő adatok előirányzott logikus sorrendjét!

```
?
?CIM [ MAJORNE ] [ FO UT ] "SOPRON "9400 "15

MAJORNE
SOPRON
9400 15
FO UT
```

2. Pontosan annyi adatot adjunk meg, amennyire szükség van!

```
?
?
?CIM [ MAJOR MARK ] [ 1011 BUDAPEST ] [ ATTILA U 33 ]
NOT ENOUGH INPUTS TO CIM
?
```

3. Egy adott paramétert betűről betűre egyformán kell használni a program során!

```
PR "
PR :NEVEK
PR :HELYSEG
( PR :UTCA :SZAM )
PR :IR.SZAM
PR "
END

?
?
?CIM [ NAGY PAL ] "1119 "BUDAPEST [ LECKE U ] "5

NEVEK HAS NO VALUE IN CIM:
PR :NEVEK
?
```

Figyeljük meg, hogy az ", ill. a szögletes zárójelnek megfelelő jeleket felváltva használhattuk, tehát akár szó, akár lista lehet a paraméter.

Ha egy sorba több bemenő adatot is ki akartunk volna írni, akkor az egész PRINT utasítássort gömbölyű zárójelbe kellett volna tennünk.

Hasonlóképpen átírható a VERS program is, pl. a Duna helyett más folyónevet írjunk ki. A PRINT bemenete legyen a SENTENCE (mondat) – ez egy új Logo funkció – és ennek bemenetei. A SENTENCE (rövidítve: SE) a bemeneteiből mondatot képez. Ez a mondat egy lista. Ha a SENTENCE-nek kettőnél több bemenete van, akkor gömbölyű zárójeleket kell használni. A SENTENCE bemenetei szavak vagy listák (mondatok) lehetnek.

```
TO VERS :FOLYO
PR ( SENTENCE :FOLYO [VIZE OLAJOS] )
PR [FURODJ BENNE ALAJOS]
END

?
?
?VERS "TISZA
TISZA VIZE OLAJOS
FURODJ BENNE ALAJOS
```

A *folyó* programváltozóként először a "TISZA" szót írjuk be. Próbáljunk meg valamilyen mondatot is. Ezzel könnyen ellenőrizhető, hogy a SENTENCE bemenetként szavakat és/vagy listákat dolgoz fel.

Befejezésül egy TEMPLOM képét fogjuk változó karakterekkel, ill. szavakkal kinyomtatni, mintegy más-más építőkövet használva.

Ehhez a WORD funkciót használjuk fel. A WORD műveletnek két vagy több bemenete van, tehát gömbölyű zárójeleket kell használni. A WORD bemenetei csak szavak lehetnek. Egyetlen betű szintén szót képez. A WORD a beírt szavakból egyetlen összefüggő jelsorozatot képez. A szóközök kinyomtatásához a CTRL-Q-t használjuk.

```
?PO "TEMPLOM
TO TEMPLOM :KO
PR "
PR ( WORD "    :KO )
PR ( WORD "  :KO "  :KO )
PR ( WORD :KO "     :KO :KO :KO )
PR ( WORD :KO "     :KO "    :KO )
PR ( WORD :KO "     :KO "     :KO )
PR ( WORD :KO "     :KO "     :KO )
PR ( WORD :KO :KO :KO :KO :KO :KO :KO :KO :KO )
END
```

Íme a program eredeti és listázott változata, majd a különböző építőelemek nyomtatási képe.

```
?TEMPLOM "$

   $
  $ $
$    $$$
$    $  $
$    $   $
$    $   $
$$$$$$$$$
?
?
?
?TEMPLOM "§ TEMPLOM "##

   §
  § §
§    §§§
§    §  §
§    §   §
§    §   §
§§§§§§§§§

   ##
  ## ##
##    ######
##    ##   ##
##    ##    ##
##    ##    ##
####################
?
?
?TEMPLOM "TEGLA

   TEGLA
  TEGLA TEGLA
TEGLA    TEGLATEGLATEGLA
TEGLA    TEGLA  TEGLA
TEGLA    TEGLA    TEGLA
TEGLA    TEGLA    TEGLA
TEGLATEGLATEGLATEGLATEGLATEGLATEGLATEGLATEGLA
?
```

Ha a változóba szó helyett egy listát írunk be, hibaüzenetet kapunk, hiszen WORD csak szó lehet.

2.3.3. Összefoglalás és gyakorlatok

1. A programokban tetszőleges funkciókat lehet hívni, azaz Logo funkciók és felhasználói funkciók egyaránt hívhatók.
2. A felhasználói funkciókat a programban úgy hívjuk, hogy a megfelelő utasítássorba beírjuk a felhasználói funkció nevét (TO nélkül!).
3. A REPEAT utasítás felhasználásával minden egyes programot előre rögzített számú alkalommal megismételhetünk.
4. A felhasználói funkciók önmagukat is hívhatják. Végeredményben ez a funkció végtelen sokszor ismétlődik.
5. Az önmagába visszatérő programot CTRL-G-vel lehet megállítani.
6. A nyomtató programok és tetszőleges funkciók tetszőleges, előre beírt adatokat dolgozhatnak fel anélkül, hogy a programot módosítanánk. Ezt is paraméterek, változók (funkcióbemenetek) segítik elő.
7. A program indításakor a programnév után egy vagy több adat (szó vagy lista) írható be. Ezek az adatok a programfejben levő előkészített változók tartalmává válnak, a további utasítássorok az előírt módon dolgozzák fel őket.
8. A paraméterek tartályokhoz hasonlíthatók, amelyeket csak a program hívásakor töltünk meg aktuális tartalommal. A program készítése során mindig ugyanúgy kell neveznünk a paramétert. A paraméter kettőspont-tal kezdődik, ezt közvetlenül követi a tetszőleges jelsorozatból álló név.
9. A TO és az ezt követő programnév (funkciónév) beírása után tetszőleges számú paraméter adható meg.
10. Amennyiben valamilyen paraméterekkel rendelkező funkció hívása során az egyikbe nem kerül tartalom, akkor a program végrehajtására nem kerül sor, hanem helyette egy hibaüzenet jelenik meg: "...NEEDS MORE INPUTS" (további bemenetekre van szükség).
11. Amennyiben a paraméter neve a programfejben és az utasítássorban eltérő, akkor a Logo jelzi, hogy e név alatt a paraméternek nincs tartalma (...HAS NO VALUE).

Új Logo szavak:

POTS, POALL
CLEARTEXT
CTRL-G
SENTENCE (SE)
(WORD...)

Feladatok

1. Írjon több versszakból álló verset!
2. Készítsen négyszög alakú keretet! Az egyes nyomtatási sorok kiíratására önálló kis felhasználói funkciókat állítson össze, amelyek csak 1-1 sort nyomtatnak ki! Az azonos sorok többszöri kinyomtatását REPEAT utasítással vezérelje!
3. Írjon saját adataival olyan címprogramot, amely önmagát hívja!
4. Készítsen NÉVJEGY néven olyan címíró programot, amely a címen kívül változóban tárolja a telefonszámot és a kinyomtatandó névjegyek számát! Használja fel erre a módosított CIMPROGRAMOT! Ezt a programot kell a CIM-ben REPEAT utasítással hívni.
5. Írjon olyan szabványos rövid levelet, amely különböző személyeket különböző rendezvényre hív meg! A programban a következő változók legyenek: a levél dátuma, címadatok, rendezvény megjelölése (meghívás)! A címzetteket a levélben „Igen tisztelt .... úr" vagy „Igen tisztelt ... asszony" megszólítással illessük! Erre a célra ugyancsak egy változót kell fenntartani.
6. „Építsen fel" egy olyan templomot, amely három különböző építőanyagból készült! Különböző építőanyagokat tegyen a torony keresztjéhez, a falazathoz és az ablakhoz.

2.4. Grafikai kirándulás

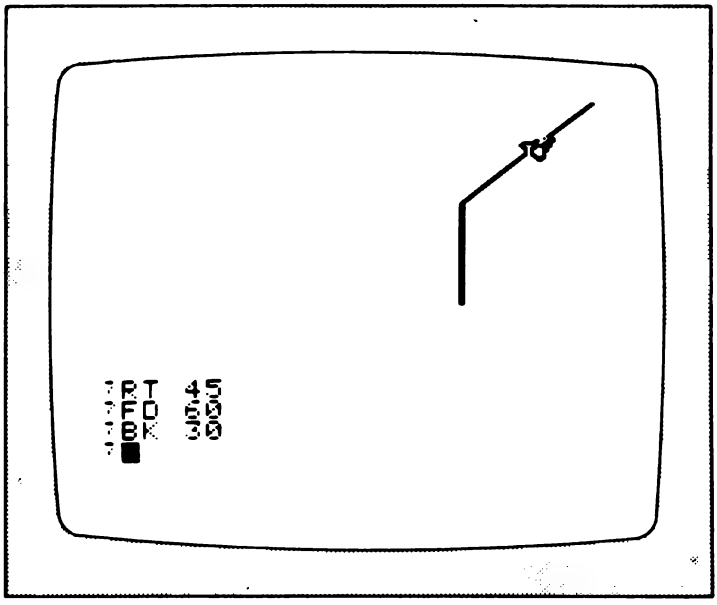
Minden Logo leírás rövid bevezetőt ad a Teknőc-grafikával. Ezek a leírások igen szépek, sok egyszerű és érdekes példát közölnek, így mi megelégszünk csupán egy grafikai fejezettel.* Még csak egy megjegyzés. A Teknőc egy háromszög alakú kurzor, ami minden irányba eltolható és tetszés szerint forgatható.

2.4.1. Séta a képernyőn

Grafikai utasítások önkényes gyűjteményét mutatjuk itt be, amit érdemes egyszerűen lemásolni. Majd meglátjuk, mi történik. Az egyes utasítások mellé egyébként rövid magyarázatokat tettünk. Bármikor kedvünk szerint térjünk el az előírt sétaúttól.

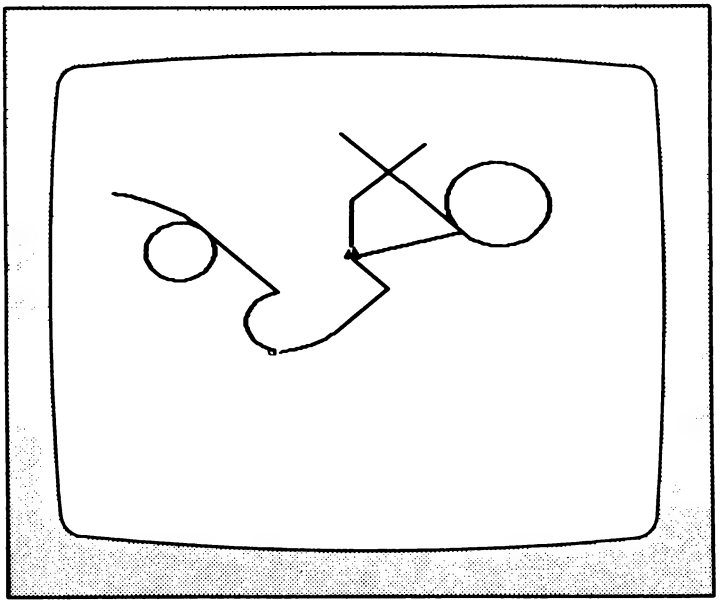
Először csak közvetlenül adjuk be a program néhány utasítását. Az FD 40 beírása után a képernyő átvált: felül van a grafikus tartomány, lent pedig négy szövegsornak a helye, aminek segítségével követni tudjuk a begépelt utasításokat, és nem teljesen tanácstalanul pötyögtetjük a billentyűket.

* E kötetben külön is összefoglaltuk a Logo grafika lehetőségeit.




```
?
?
?TO SETA
>FD 40          előre 40 lépés (pixel)
>RT 45          jobbra 45° fordulás
>FD 60          előre 60 lépés
>BK 30          vissza 30 lépés
>LT 90          balra 90° fordulás
>FD 40          előre 40 lépés
>BK 100         vissza 100 lépés
>CIRCLER 30     30 lépés sugarú kör
>SETPOS A0 0U   Az X=0, Y=0 pontra ugrani
>BK 30          vissza 30 lépés
>LT 90          balra 90° fordulás
>FD 40          előre 40 lépés
>ARCR 50 40     50 lépés sugarú 40°-os körív jobbra
>ARCR 20 180    20 lépés sugarú 180°-os körív jobbra
>LT 130         balra 130° fordulás
>FD 60          előre 60 lépés
>CIRCLEL 20
>ARCL 100 40    100 lépés sugarú 40°-os körív balra
>PU             tollat fel
>HOME           kezdő pontra (haza)
>PD             tollat le
>END
SETA DEFINED
?
```

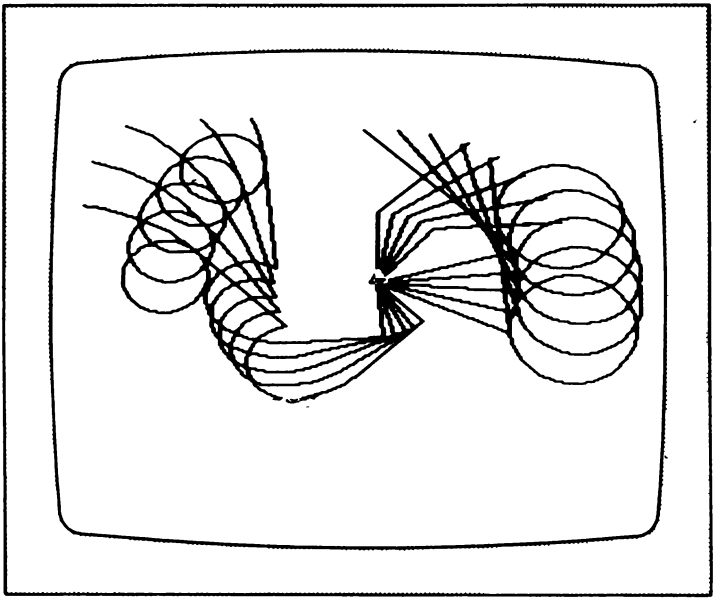
Utólag még illesszük be a program elejére a CLEARSCREEN (CS) utasítást.



A HOME utasítás a teknősbékát visszaviszi a kiindulási helyzetbe. Ekkor a teknősbéka egyenesen felfelé néz. Fordítsuk el a teknősbékát 10°-kal jobbra, és hívjuk ismét a SETA-t. Ezt követően forgassuk el további 10°-kal a teknősbékát stb.

```
?RT 10 SETA
?RT 20 SETA
?RT 30 SETA
?RT 40 SETA
?RT 50 SETA
?
```

A képernyőn ekkor a következő látható:



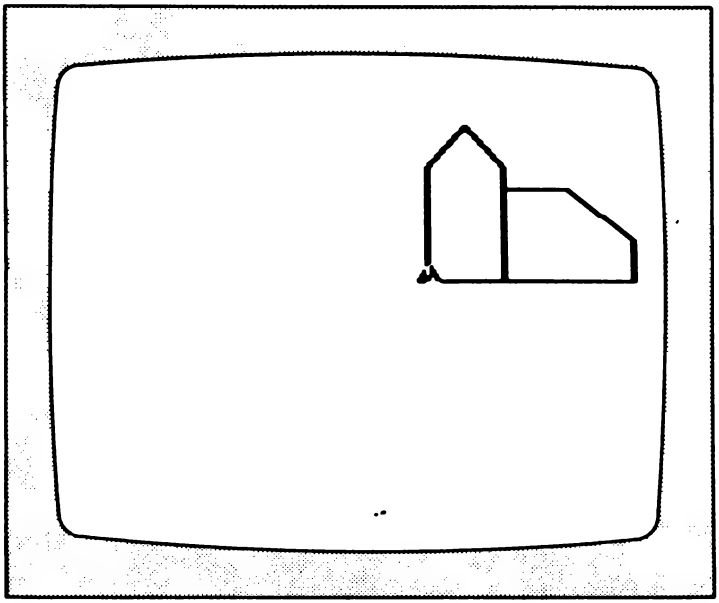
A szövegsorok környékén a rajz egy része tönkremegy. Ezért a FULLSCREEN utasítással gondoskodtunk arról, hogy a teljes kép látható legyen.

2.4.2. Nagyítás és kicsinyítés

A bemutatott utasítások segítségével bármikor tetszőleges ábrák rajzolhatók a képernyőre. Némi fantáziával geometriai ábrákból és görbékből a világon minden lerajzolható. A következőkben egy tornyot és egy templomhajót mutatunk be.

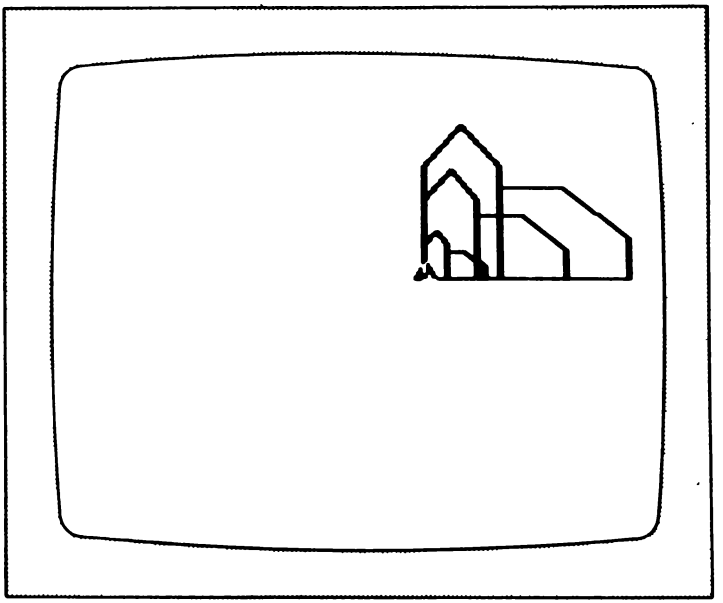
```
?
?TO TEMPLOM
>TORONY
>HAJO
>END
TEMPLOM DEFINED
?
?TO TORONY
>FD 70
```

```
>RT 30
>FD 30
>RT 120
>FD 30
>RT 30
>FD 70
>HOME
>END
TORONY DEFINED
?
?
?TO HAJU
>RT 90
>FD 30
>LT 90
>FD 56
>RT 90
>FD 25
>RT 50
>FD 40
>RT 40
>FD 25
>HOME
>END
HAJO DEFINED
?
```



Még érdekesebb ábrákat kapunk, ha felhasználjuk a funkcióbemenetekkel kapcsolatos tudományunkat, és az egyes grafikai utasítások számérétkeit tetszés szerint változtatjuk. A legvadabb lehetőségek nyílnak meg. A következőkben csak az egyenes mozgásokat változtatjuk, a szögadatokat állandó értéken tartjuk. Kis fáradsággal megoldható a dolog, a jelenlegi számot megszorozzuk egy változóval:

```
  ?TO TEMPLOM :MERET
  >TORONY1 :MERET
  >HAJO1 :MERET
  >END
  TEMPLOM DEFINED
  ?

?
?TO HAJO1 :MERET
>RT 90
>FD 30 * :MERET
>LT 90
>FD 56 * :MERET
>RT 90
>FD 25 * :MERET
>RT 50
>FD 40 * :MERET
>RT 40
>FD 25 * :MERET
>HOME
>END
HAJO1 DEFINED
?
?
?TO TORONY1 :MERET
>FD 70 * :MERET
>RT 30
>FD 30 * :MERET
>RT 120
>FD 30 * :MERET
>RT 30
>FD 70 * :MERET
>HOME
>END
TORONY1 DEFINED
?
?
```

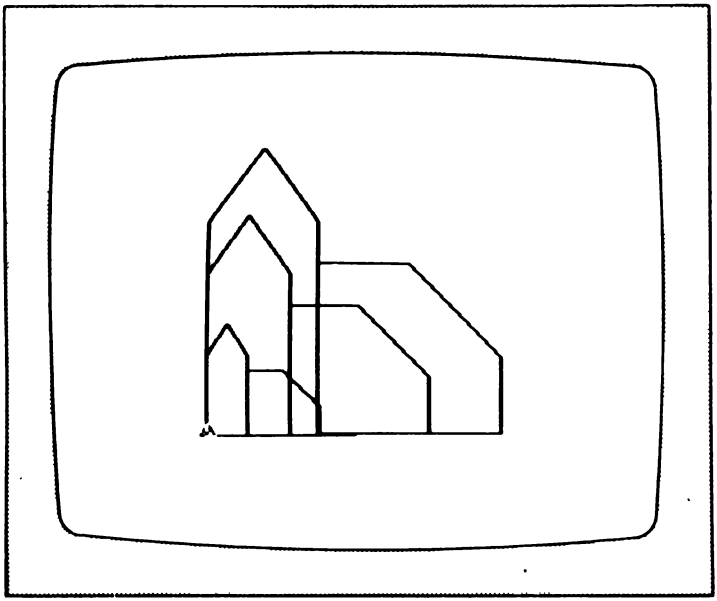


Ha a méret bemenete 2, akkor kilépünk a képernyőről. A Teknőc kiindulási pontját tehát balra, alulra kellene helyeznünk. Az erre szolgáló funkciót így definiálhatjuk:

```
?
?TO KEZDOPONT :POS
>SETPOS. :POS
>SETH 360
>END
KEZDOPONT DEFINED
?
```

SETH rövidítve SETHEADING-et jelent, azaz a Teknőc a megadott égtáj felé néz. Az égtájat fokban, az iránytű szerint adjuk meg. Ebben az esetben a Teknőcnek pontosan észak felé kell néznie.

Teszteljük a KEZDOPONT-ot különböző értékekkel. A bemenet minden esetben a koordináták listája. A képernyő középpontja [00] · [–100 –95] jó érték. A képernyőbeosztás a felhasználói kézikönyvben található. Építsük be a kezdőpontot a programba!



```
TO TORONY1 :MERET
FD 70 * :MERET
RT 30
FD 30 * :MERET
RT 120
FD 30 * :MERET
RT 30
FD 70 * :MERET
KEZDOPONT :POS
END

TO HAJO1 :MERET
RT 90
FD 30 * :MERET
LT 90
FD 56 * :MERET
RT 90
FD 25 * :MERET
RT 50
FD 40 * :MERET
RT 40
FD 25 * :MERET
KEZDOPONT :POS
END

TO TEMPLOM :MERET :POS
PU KEZDOPONT :POS PD
TORONY1 :MERET
HAJO1 :MERET
END
```

A TEMPLOM első utasítássora talán kissé meglepő. De nyilvánvaló, hogy a kiindulási pont felé haladva a Teknőcnek nem maradhat nyoma. Ezért a tollat először fel kell emelnünk, majd pedig az adott hely elérése után ismét le kell tennünk. Különböző értékekkel kipróbálhatjuk, hogy mi a helyzet, ha a kiindulási pontot változtatjuk.

```
?
?TEMPLOM .6 [30 -100]
?
```

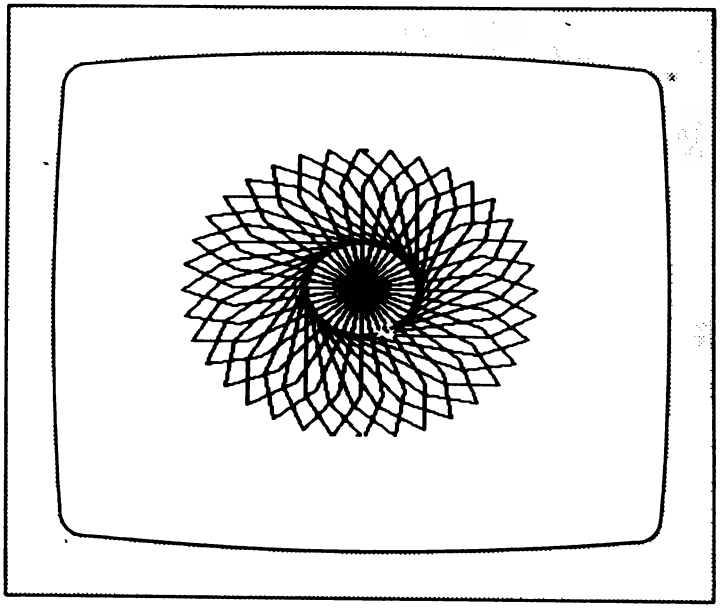
2.4.3. Csillagköd

Gyönyörű eredményhez juthatunk, ha a rajzokat megforgatjuk. Erre már a 2.4.1. szakaszban utaltunk. Készítsünk egy ilyen forgatóprogramot és pörgessük meg pl. a templomtornyot. A legegyszerűbb módszer egy olyan programváltozat, amelyben a program örökké önmagát hívja. A TORONY1 utolsó sorába írjuk be a HOME-ot.

```
?
?TO FORGAS :FOK :FAKTOR
>TORONY1 1
>RT :FOK * :FAKTOR
>FORGAS :FOK :FAKTOR + 1
>END
FORGAS DEFINED
?
```

A :FOK változó adja meg az egyes rajzolási műveletek utáni szögelfordulást. A :FAKTOR módosítja a FORGAS mértékét, ismételt felhívásakor a :FOK pl. a 15, 30, 45 ... értékeket veheti fel.

```
?
?CLEARSCREEN
?FORGAS 15 1
```



2.4.4. Összefoglalás és gyakorlatok

1. A Teknőc a képernyőn minden irányba mozoghat. Balra vagy jobbra forgatható, előre vagy hátra halad, a képernyő tetszőleges pontjára helyezhető ... A Teknőcöt szimbolikusan egy háromszög jelképezi. A háromszög csúcsa az előremutató irányt jelzi, előrehaladási utasítás esetén a Teknőc erre indul.
2. A Teknőc minden mozgásakor nyomot hagy. A nyom rajzolása letiltható. Erre akkor van szükség, ha a képernyőn különálló ábrákat akarunk készíteni.
3. Ha többször dolgozunk grafikai utasításokkal, akkor célszerű előzetesen törölni a képernyőt (CLEARSCREEN).
4. A Teknőccel folytatott munka során a képernyő két részre van osztva. A képernyő alsó részén négy begépelt utasítássor számára van hely. Amennyiben a teljes képernyőt a grafika számára kívánjuk fenntartani, akkor be kell írnunk a FULLSCREEN utasítást.
5. Az irányváltoztatásokon, valamint az előre- és hátramozgáson kívül a Teknőc köríveket és teljes köröket is tud rajzolni.
6. Tetszőleges grafikai utasításokból program készíthető, ezzel saját funkciók

hozhatók létre. A kísérletezés során tanácsos a grafikai utasítások bemeneteként változókat megadni.

7. Érdekes hatások érhetők el, ha a rajzok és ábrák méretét megváltoztatjuk, vagy a képernyőn több ábrát helyezünk el, vagy pedig az ábrákat elforgatjuk.
8. TEXTSCREEN segítségével elhagyhatjuk a grafikai üzemmódot, ekkor visszakapjuk a szokásos képernyőbeosztást.

Új Logo szavak:

FORWARD (FD)	HOME
RIGHT (RT)	PENDOWN (PD)
BACK (BK)	CLEARSCREEN (CS)
LEFT (LT)	FULLSCREEN
CIRCLER	SETHEADING (SETH)
ARCR	SETPOS
CIRCLEL	TEXTSCREEN
ARCL	SETSCRUNCH
PENUP (PU)	(a gyakorlatokban)

Feladatok

1. Írjon olyan programot, amely változó oldalhosszúságú négyzeteket rajzol!
2. Írjon olyan programot, amely változó nagyságú, egyenlő oldalú háromszögeket rajzol!
3. Rajzoltasson különböző nagyságú fekvő nyolcasokat!
4. Rajzoljon változó körszeleteket (tortaszeleteket)!
5. Az eddig előállított elemekből készítsen „pont, pont, vesszőcske" emberfejet!
6. Az eddigi programok felhasználásával a képernyő különböző pontjaira rajzoljon különböző méretű ábrákat!
7. Hány lépést tehet a Teknőc a középpontból felfelé, lefelé, balra és jobbra a képernyő széléig?
8. A SETSCRUNCH utasítás a képernyőn levő ábrákat széthúzza vagy összenyomja. A vízszintes lépés–függőleges lépés leképezési arányt a SETSCRUNCH állítja be. A SETSCRUNCH után beírt szám általában az adott képernyőtől függően a 0.8, ..., 1.2 tartományban van, így elérhető, hogy a kör valóban kerek legyen. E hatás tanulmányozására kísérletezzen a SETSCRUNCH-csal és különböző módosított értékekkel! Négyzetekből téglalapok, körökből ellipszisek lesznek stb.

9. Módosítsa az 1. és 2. feladatot úgy, hogy a geometriai ábra ne záródjon, minden egyes rajzolandó oldal egy kicsivel legyen hosszabb, a forgatás azonban ne változzon! (Útmutatás: ez legegyszerűbben úgy oldható meg, hogy a program változó bemenettel önmagát hívja.)
10. Az előbbi feladatokban előállított ábrákat forgassa a 2.4.3. szakaszban ismertetett FORGAS programmal!

2.5. Nyomtatót használunk, a programokat mágneslemezen tároljuk

Munkánk eredményét papírra szeretnénk nyomtatni, egyes programokat mágneslemezen (diskette) szeretnénk tárolni annak érdekében, hogy későbbi időpontban ezeket a programokat ismét betölthessük a tárba, és megint használhassuk őket.

A nyomtató a következő utasítással helyezhető üzemkész állapotba:

```
.PRINTER 1.
```

Minden, amit beírunk, és minden adatkiíratás kinyomtatódik. A nyomtató kikapcsolása:

```
.PRINTER 0.
```

A nyomtató írásszélessége meghaladhatja a soronkénti 40 jelet. Ez kényelmes, mivel egyes Logo sorok hosszabbak, mint egy képernyősor. Az Apple mátrixnyomtatójánál a nyomtatót úgy állítják be soronként 80 jelre, hogy a nyomtató üzembehelyezése után a következő utasítást írjuk be:

```
PR WORD CHAR 9 "80N.
```

Ily módon a nyomtatónak vezérlőjeleket adunk, de ezzel itt most tovább nem foglalkozunk.

A teljes munkatárat mágneslemezen tárolhatjuk. Ezt a SAVE utasítás végzi el. Már csak egy nevet kell kitalálnunk a tárolt program számára, ami lehetővé teszi, hogy a mágneslemezen megfelelő módon katalogizálva legyen:

```
SAVE "LAJOS.
```

Írjuk most be a CATALOG utasítást, erre a gép kiírja a mágneslemez tartalomjegyzékét, a "LAJOS tételnek a tartalomjegyzékben szerepelnie kellene.

LOAD utasítással a mágneslemez tartalmát beolvastatjuk a számítógépbe:

```
LOAD "LAJOS.
```

A fájlokat a mágneslemezen az ERASEFILE file utasítással töröljük.

```
ERASEFILE "LAJOS
```

Ha már van LAJOS nevű fájl, akkor ugyanilyen néven még egyet nem menthetünk ki. Tehát a régi fájlok védve vannak. Csak előzetes ERASEFILE utasítással lehet már meglevő fájl néven új fájlt kimenteni.

2.6. Műveletek definiálása

2.6.1. Kimenet: OUTPUT

Már egy sereg Logo műveletet ismerünk. Közéjük tartozik, a FIRST, BUTFIRST, EMPTYP, COUNT és néhány számítási és összehasonlító művelet. Az 1.3. szakaszban a műveleteket egymáshoz kapcsoltuk. Az utasítással ellentétben, egy műveletnek mindig van eredménye. A művelet által adott eredmény egy utasítás vagy egy másik művelet bemenete. A következő példát már ismerjük:

```
?PRINT FIRST BF BF [EN TE O MI TI OK]
O
?
```

Ez az utasítássor egy lista harmadik elemét nyomtatja ki. Az utasítássor köré írhatunk egy kis nyomtató programot.

```
TO NY.HARMADIK :LISTA
PR FIRST BF BF :LISTA
END

?
?NY.HARMADIK [EN TE O MI TI OK]
O
?
```

A program ily módon egy felhasználó által definiált utasítás, mivel ebben PRINT utasítás is van. De a NY.HARMADIK még így nem művelet. A lista általa meghatározott harmadik elemét nem adja tovább, hanem kinyomtatja. PRINT helyett más Logo funkciót kell használnunk, ha műveletet akarunk kapni: ez pedig az OUTPUT.

Próbáljuk ki a HARMADIK nevű programot!

```
?
?
?TO HARMADIK :LISTA
>OUTPUT FIRST BF BF :LISTA
>END
HARMADIK DEFINED
?
?PR HARMADIK [EN TE O MI TI OK]
O
```

```
?
?
?PR FIRST HARMADIK [ EN TE OK ]
O
?
?
?HARMADIK [ EN TE OK ]
I DON'T KNOW WHAT TO DO WITH OK
?
```

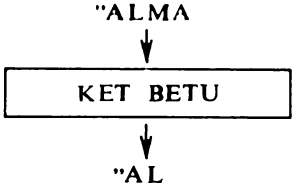
A példák alapján látjuk, hogy a HARMADIK egy művelet, amely más Logo műveletekhez láncolható. Ha a HARMADIK által szolgáltatott eredmény nem bemenete egy utasításnak, akkor megjelenik a szokásos hibaüzenet: "I DON'T KNOW WHAT TO DO WITH ...".

A következőkben definiálunk egy második felhasználói műveletet és megvizsgáljuk, hogy az általunk definiált műveletek egymással láncolhatók-e. Készítsünk olyan műveletet, amely a bemenet első három elemét eltávolítja!

```
?
?
?TO 3EL :TARGY
>OP BF BF BF :TARGY
>END
3EL DEFINED
?
?
?PR 3EL "MISKOLC
KOLC
?
?
?PR 3EL HARMADIK [ MIKOR MEGYEK MISKOLCRA ]
KOLCRA
?
?
?PR COUNT 3EL [ M I S K O L C ]
4
?
?PR BF BF 3EL HARMADIK [ D F ABRAKADABRA ]
ADABRA
?
?
```

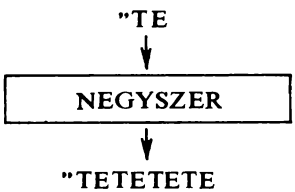
A következőkben feladatként bemutatunk néhány műveletet. A feladat mellett egy példa segítségével vázlatosan tárgyaljuk a műveletet. Ezt követően minden egyes műveletet kilistázunk és tesztelünk.

1. Írjon egy olyan műveletet, amely a beírt szó első két szavát adja meg!



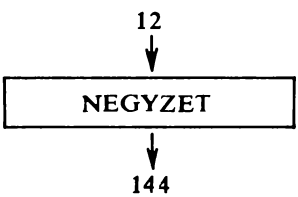

```
?
?TO KETBETU :SZO
>OP WORD FIRST :SZO FIRST BF :SZO
>END
KETBETU DEFINED
?
?PR KETBETU "ALMA
AL
```

2. Írjon egy olyan műveletet, amely a beírt szót megnégyszerezi!



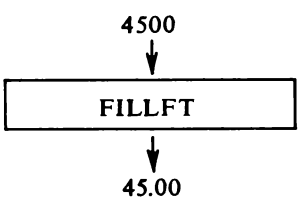

```
?TO NEGYSZER :SZO
>OP (WORD :SZO :SZO :SZO :SZO)
>END
NEGYSZER DEFINED
?
?
?PR NEGYSZER "TE
TETETETE
?
```

3. Írjon egy olyan műveletet, amely a beírt számot négyzetre emeli!



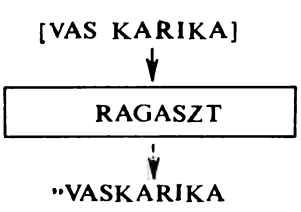

```
?
?TO NEGYZET :SZAM
>OP :SZAM * :SZAM
>END
NEGYZET DEFINED
?
?
?PR NEGYZET 12
144
?
```

4. Írjon egy olyan műveletet, amely a beírt fillérösszeget forinttá számítja át!



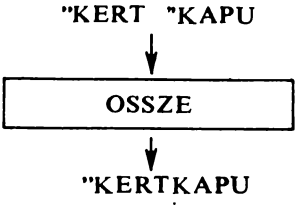

```
TO FILLFT :FILL
OP ( WORD BL BL :FILL ". LAST BL :FILL LAST :FILL )
END

?
?
?PR FILLFT 12345
123.45
?
```

5. Írjon egy olyan műveletet, amely a kételemű lista szavait egy szóvá illeszti össze!



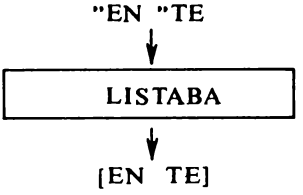

```
?
?TO RAGASZT :LISTA
>OP WORD FIRST :LISTA LAST :LISTA
>END
RAGASZT DEFINED
?
?PR RAGASZT [VAS KARIKA]
VASKARIKA
?
```

6. Írjon egy olyan műveletet, amely a két beírt szót egy szóvá illeszti össze!



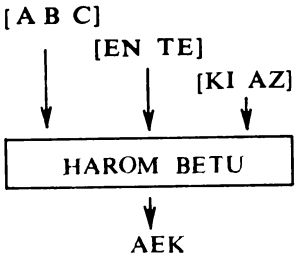

```
?
?TO OSSZE :SZO1 :SZO2
>OP WORD :SZO1 :SZO2
>END
OSSZE DEFINED
?
?PR OSSZE "KERT "KAPU
KERTKAPU
?
```

7. Írjon egy olyan műveletet, amely a két beírt szót kételemű listává alakítja!




```
?
?TO LISTABA :SZO1 :SZO2
>OP SENTENCE :SZO1  :SZO2
>END
LISTABA DEFINED
?
?PR LISTABA "EN "TE
EN TE
?PR LISTP LISTABA "EN "TE
TRUE
?PR COUNT LISTABA "EN "TE
2
?
```

8. Írjon egy olyan műveletet, amely a beírt három listából minden egyes szó első betűjét egy szóvá illeszti össze!




```
TO 3BETU :L1 :L2 :L3
OP ( WORD FIRST FIRST :L1 FIRST FIRST :L2 FIRST
FIRST :L3 )
END
?
?PR 3BETU [A B C] [EN TE] [KI AZ]
AEK
?
?
```

Mint már említettük, természetesen az önmagunk által definiált műveleteket tetszőlegesen láncolhatjuk egymáshoz. Ez csak addig eredményes, amíg az összetétel ésszerű és az egyes műveletek bemenete a definíciónak megfelelő. A NEGYSZER bemenete nem lehet lista. A KÉTBETU bemenetének legalább kétbetűs szónak kell lennie stb. Lássunk néhány lehetséges láncolást. Befejezésül néhány hibát ejtettünk, hogy megnézhessük a jellegzetes hibaüzeneteket.

```
?
?PR NEGYZET NEGYZET NEGYZET 12
429981696
?
?
?PR OSSZE "HAZ NEGYZET 12
HAZ144
?
?
?PR NEGYSZER KETBETU OSSZE NEGYZET 3 NEGYZET 12
91919191
?
?
?PR RAGASZT []
FIRST DOESN'T LIKE [] AS INPUT:
 JUST BEFORE LEAVING RAGASZT
?
?
?PR KETBETU [SZO]
```

```
FIRST DOESN'T LIKE [] AS INPUT:
 JUST BEFORE LEAVING KETBETU
?
?PR KETBETU "A
FIRST DOESN'T LIKE  AS INPUT:
 JUST BEFORE LEAVING KETBETU
?
```

2.6.2. Választás: ITEM

Az előző szakaszban megismerkedtünk a HARMADIK művelettel. Hasonló módon műveleteket készíthetünk, egy lista negyedik, hatodik vagy kilencedik elemének meghatározására. Erre szolgál az ITEM nevű Logo művelet. Az ITEM műveletben pontosan meg kell mondanunk, hogy melyik lista melyik elemére van szükségünk. Először a megfelelő számot, utána pedig a listát írjuk be.

```
?
?PR ITEM 3 [ KI BE FEL LE ]
FEL
?
```

Ezt az ITEM dolgot gyakorlásképpen becsomagoljuk az N.ELEM nevű műveletbe. Egyébként ez ugyanazt csinálja, mint az ITEM. Tulajdonképpen csak magyarítottuk.

```
?
?TO N.ELEM :MELYIK :LISTA
>OP ITEM :MELYIK :LISTA
>END
N.ELEM DEFINED
?
?PR N.ELEM 2 [ KI BE FEL LE ]
BE
?
```

2.6.3. Véletlen: RANDOM

A véletlenszámok előállítására a RANDOM (véletlen) nevű funkció szolgál. A RANDOM Logo művelet bemenete mindig egy szám. Ez a szám megadja azt a tartományt, amelybe a véletlenszámoknak esniük kell.

```
?
?PR RANDOM 5
2
?
?REPEAT 20 [ TYPE RANDOM 5 ]
21212324343124032320?
?
```

Látjuk tehát, hogy az 5 bemenet hatására ötféle véletlenszám jelenhet meg, a 0, 1, 2, 3 és 4. A határ már nem megengedett.

Készítsünk olyan dobókockát, amely az 1...6 pontszámokat adhatja. A nullát valamiképpen meg kell tiltanunk. Ha pl. az előbbi példában mindig hozzáadtunk volna az eredményhez egyet, akkor 1...5 közötti véletlenszámokat kaptunk volna. Ezt az apró trükköt alkalmazzuk a kocka esetében is:

```
TO KOCKA
OP 1 + RANDOM 6
END

?
?REPEAT 20 [ TYPE KOCKA ]
5413415532355166666 5?
?
```

Amennyiben három kockával való dobást akarunk szimulálni, akkor KOCKA három dobásának eredményét össze kell adnunk, vagyis:

```
?
?
?TO HAROMDOBAS
>OP (SUM KOCKA KOCKA KOCKA)
>END
HAROMDOBAS DEFINED
?
?REPEAT 10 [ TYPE HAROMDOBAS TYPE "  ]
? 11 8 12 9 9 13 10 13 6 ?
?
?
```

Természetesen véletlen betűket vagy szavakat is előállíthatunk, ha számokat képezünk és a számítógéppel meghatározhatjuk, hogy ezen kódszámokhoz milyen betűk tartoznak. Fellapozva a kézikönyvet, megállapíthatjuk az A...J betűk kódszámait:

A – 65;
B – 66;
C – 67;
..... ;
I – 73;
J – 74.

Milyen műveleteket kell írnunk, ha ötbetűs szavakat akarunk gyártani, amelyek csak A...J betűkből állnak? Először 65...74 tartományba eső kódszámoknak megfelelő véletlenszámoket kell generálni. Majd egy művelettel meg kell határozni a hozzá tartozó betűket. Ezt könnyen megoldja a CHAR (jelet vagy betűt jelent) Logo művelet. Végül öt betűből egy szót kell összeállítani.

Próbáljuk meg!

2.6.4. Véletlenregény

A számítógép ugyan verseket vagy regényeket nem tud írni, de mondatrészek előkészített készletéből tetszőleges kombinációkat képez. Készítsünk egy ilyen programot!

Állapodjunk meg egyelőre abban, hogy a számítógép 7 alanyból, 7 állítmányból stb. álló készletből gazdálkodik. Ebből a négyszer 7 mondatrészből tevődnek össze a mondatok, amelyek természetesen azonos szerkezetűek. Ily módon újra és újra előállítunk akár 7^4 különböző mondatot. A REPEAT utasítással ez könnyen megoldható:

```
?
?TO REGENY :SOROK.SZAMA
>REPEAT :SOROK.SZAMA [ PR VMONDAT ]
>END
REGENY DEFINED
?
```

A VELETLENMONDAT művelet az előbbi szabály szerint egyetlen mondatot állít össze:

```
?
?TO VMONDAT
>OP (SE ALANY HELYH MODH ALLITMANY)
>END
VMONDAT DEFINED
?
```

Először az ALANY műveletet állítsuk elő, amelynek 7 lehetőségnyi készletből 1 alanyt választ. Az N. ELEM és a VELETLEN műveleteket már ismerjük. A VELETLEN 1 és 7 közé eső véletlenszámot ad. Az ALANYLISTA adja a 7 alanyt, a játékanyagot.

A MODHATAROZO, HELYHATAROZO és ALLITMANY műveletek definíciója hasonló. Ily módon a következőkben kilistázott, általunk definiált funkciók adódnak:

```
?
?TO ALANY
>OP N.ELEM VELETLEN ALANYLISTA
>END
ALANY DEFINED
?
?TO HELYH
>OP N.ELEM VELETLEN HELYHLISTA
>END
HELYH DEFINED
?
?
```

```
?TO MODH
>OP N.ELEM VELETLEN MODHLISTA
>END
MODH DEFINED
?
?TO ALLITMANY
>OP N.ELEM VELETLEN ALLITMANYLISTA
>END
ALLITMANY DEFINED
?
TO ALANYLISTA
OP [HALACSKA AZELNOK KARCSI FIFI NEMZETIMUZEUM
LUJZA BOSZORKA]
END
?
?TO ALANYLISTA
>OP [ TE MI KARCSI FIFI NEMZETIMUZEUM LUJZA
>BOSZORKA]
>END
ALANYLISTA DEFINED
?
?
?TO HELYHLISTA
>OP [ MOZIBAN SZINHAZBAN ITT ISKOLABAN KERTBEN
>FAN FALUN ]
>END
HELYHLISTA DEFINED
?
?TO MODHLISTA
>OP [ JOZANUL HANYAGUL BUSAN VIDAMAN VADUL UGRALVA
>REBEGVE ]
>END
MODHLISTA DEFINED
?
?
?TO ALLITMANYLISTA
>OP [ FUTYUL ALSZIK ALL SENYVED KERES ESZIK
>CSIPEGET ]
>END
ALLITMANYLISTA DEFINED
?
?
?TO VELETLEN
>OP SUM 1 RANDOM 7
>END
VELETLEN DEFINED
?
```

Mielőtt elindítanánk a programot, még gyorsan néhány tanács. A programok sohasem hibátlanok. A Logo rendelkezik azzal a kényelmes lehetőséggel, hogy minden egyes felhasználói funkciót – akár utasítás az, akár művelet – külön ki lehet próbálni. Mivel tudjuk, hogy az egyes funkcióknak mit kellene csinálniuk, egyenként kipróbálgathatjuk őket. A következőkben néhány példát mutatunk arra, hogy miképpen végezhetnénk el ezt, ha a hibaüzenetek

az egyik általunk definiált funkcióra utalnának. Jobb azonban minden esetben előzetesen elvégezni az egyedi ellenőrzést. Különösen a mondatrészlisták veszélyesek, hiszen lehet, hogy elszámoltuk magunkat. Listánként ui. 7 elemnek kell lennie. A COUNT segítségével ez könnyen ellenőrizhető.

Ha már többé kevésbé biztosak vagyunk abban, hogy minden rendben van, akkor próbáljuk ki a regényt.

```
?
?
?PR REGENY 4
NEMZETIMUZEUM ISKOLABAN VADUL SENYVED
LUJZA FALUN VADUL ESZIK
FIFI FALUN JOZANUL KERES
FIFI KERTBEN BUSAN ALSZIK
REGENY DIDN'T OUTPUT TO PRINT
?
```

2.6.5. Összefoglalás és gyakorlatok

1. A felhasználó által definiált műveletek éppúgy működnek, mint a Logo rendszer műveletei. A felhasználó által definiált műveletek a Logo utasítások vagy más műveletek bemenetei lehetnek.
2. A felhasználó által definiált műveletek tetszőlegesen láncolhatók más felhasználói műveletekkel vagy a Logo rendszer műveleteivel.
3. A felhasználói funkció attól válik műveletté, hogy a feldolgozás eredményét az OUTPUT továbbítja. Ezzel a művelet futása befejeződik.

Új Logo szavak:

OUTPUT
ITEM
RANDOM
CHAR

Feladatok

1. Definiáljon olyan műveletet, amelyik a téglalap területét adja!
2. Készítsen olyan műveletet, amely megadja a két szám átlagát!
3. Definiáljon olyan műveletet, amelyik felcseréli egy kételemű lista elemeit!
4. Készítsen olyan műveletet, amely a melléknév középfokából a felsőfokot állítja elő!
5. Készítsen olyan műveletet, amely valamely lista utolsó három elemének (szavának) utolsó betűiből listát képez!

6. Készítsen olyan műveletet, amely előállít öt lottószámot (1 és 90 közé eső szám)!
7. Készítsen olyan műveletet, amelyik ötjegyű véletlenszámot képez!
8. Készítsen olyan műveletet, amely tetszőleges másodfokú polinomra ($y = Ax^2 + Bx + C$) megadja az y értékét x és a paraméterek függvényében!

2.7. Igaz és hamis vizsgáló szavak

2.7.1. Vizsgáló szavak a szókészletben

A Logoban lehetőség van arra, hogy a programon belül ellenőriztessük az adatokat. Egy ilyen vizsgálat eredménye minden esetben vagy "TRUE vagy "FALSE (igaz vagy hamis).

```
?
?PR EMPTYP "TANGO
FALSE
?PR EMPTYP [A EZ A LISTA ]
FALSE
?
?PR EMPTYP []
TRUE
?
?PR EMPTYP BF BF BF "TEA
TRUE
?
```

Az EMPTYP megvizsgálja, hogy a bemenet üres szó vagy üres lista-e. A Logo vizsgáló szavait arról ismerjük meg, hogy P-vel fejeződnek be (Proof, vizsgálat).

Példák segítségével ismerkedjünk meg néhány Logo vizsgáló szóval; további magyarázatra bizonyára nincs szükség.

```
?
?
?PR WORDP "SZO
TRUE
?PR WORDP [SZO]
FALSE
?PR WORDP "176
TRUE
?PR WORDP 176
TRUE
?
?
?PR NUMBERP [123]
FALSE
```

```
?PR NUMBERP FIRST [123]
TRUE
?PR LISTP []
TRUE
?PR LISTP "LISTA
FALSE
?PR LISTP FIRST [123]
FALSE
?
?
?
?PR EQUALP "EN "TE
FALSE
?PR EQUALP "EN "EN
TRUE
?PR EQUALP "EN [EN]
FALSE
?PR EQUALP [A] FIRST [A B] [A]
TRUE
?PR EQUALP 3 5
FALSE
?PR 3=5
FALSE
?
?PR MEMBERP "A [A B E L]
TRUE
?PR MEMBERP "VAN [MINDEN RENDBEN VAN]
TRUE
?PR MEMBERP "VAK [MINDEN RENDBEN VAN]
FALSE
```

A WORDP vizsgáló szó esetében a 176 számot egyszer az előírásos idézőjellel, egyszer anélkül írtuk be. Ez csak számoknál szabályos. A számok szavak, de a szó nem feltétlenül szám, mint ahogy ezt a NUMBERP-pel bemutatott példák is mutatják. Az EQUALP vizsgáló szó rövid alakja a „=". A MEMBERP vizsgáló szó második bemenetének listának kell lennie. E vizsgáló szó segítségével tehát nem tudjuk azt megvizsgálni, hogy egy betű előfordul-e egy szóban. Ilyen vizsgáló műveletet tehát magunknak kell készíteni.

Egyes esetekben hasznos, ha több vizsgálati eredményt egymáshoz kapcsolunk. Szükségünk lehet pl. egy olyan vizsgálatra, amelynél három feltételnek kell teljesülnie ("TRUE). Előfordulhat az is, hogy több lehetőség közül legalább egynek kell teljesülnie. Az ilyen „Igazságok" összekapcsolása a BOTH vagy OR műveletekkel történik. A "TRUE-t vagy "FALSE-t az ellenkezőjére fordíthatjuk, a NOT művelet segítségével. Példák alapján lássunk ilyen kapcsolásokat:

```
?
?PR NOT LISTP [EN TE O]
FALSE
?PR NOT MEMBERP "A [A B C]
FALSE
```

```
?PR AND 3<4 6>4
TRUE
?
?
?PR OR 3<4 4>6
TRUE
?PR OR WORDP [A] LISTP [LAP]
TRUE
?PR NOT OR WORDP [A] LISTP [LAP]
FALSE
?
```

2.7.2. Felhasználó által definiált vizsgáló szavak

A Logoban mindent, amire szükségünk van, saját funkcióként (felhasználó által definiált funkcióként) állíthatunk elő. Erre szolgálnak pl. a felhasználó által definiált vizsgáló szavak. Minden ilyen vizsgáló szó után tegyünk kérdőjelet. Aki akar, természetesen maradhat a hagyományos P betűnél.

A következőkben bemutatunk néhány mintafeladatot, rögtön kilistázzuk és teszteljük őket.

1. Készítsen olyan vizsgáló szót, amely megvizsgálja, hogy a beírt szám nagyobb-e 5-nél!

```
?
?TO NA5? :SZAM
>OP :SZAM > 5
>END
NA5? DEFINED
?
?PR NA5? 10
TRUE
?PR NA5? 4
FALSE
?
```

2. Készítsen olyan vizsgáló szót, amely megállapítja, hogy valamely bemenetlista öt listaelemből áll-e!

```
?
?TO 5OSLISTA :LISTA
>OP 5=COUNT :LISTA
>END
5OSLISTA DEFINED
?
?PR 5OSLISTA [A B CEE EN TE E]
TRUE
?PR 5OSLISTA [A B CEE]
FALSE
?PR 5OSLISTA []
FALSE
?
```

3. Készítsen olyan vizsgáló szót, amely a két beírt listát megvizsgálja abból a szempontból, hogy az első listának több eleme van-e, mint a másodiknak!

```
TO HOSSZABB? :LISTA1 :LISTA2
OP COUNT :LISTA1 > COUNT :LISTA2
END

?
?PR HOSSZABB? [A B C D] [EN]
TRUE
?PR HOSSZABB? [EN] [ABCD]
FALSE
?PR HOSSZABB? [A] [B]
TRUE
?
```

4. Készítsen olyan vizsgáló szót, amely azt vizsgálja, hogy a beírt szám kisebb-e nullánál!

```
?
?
?TO NEGATIV :SZAM
>OP :SZAM<0
>END
NEGATIV DEFINED
?
?PR NEGATIV 123
FALSE
?PR NEGATIV -12
TRUE
?
```

5. Készítsen olyan vizsgáló szót, amely megállapítja, hogy a két beírt betű betűrendben van-e! Az első betű álljon a második betű előtt! (Útmutatás: Az ASCII művelettel a betű kódtáblázat szerinti sorszámát kapjuk. L. még a 2.6.3. szakaszt!)

```
?
?TO ELOTT? :BETU1 :BETU2
>OP ASCII :BETU1<ASCII :BETU2
>END
ELOTT? DEFINED
?
?PR ELOTT? "A "B
TRUE
?PR ELOTT? "B "A
FALSE
?PR ELOTT? "B "Z
TRUE
?
```

6. Készítsen olyan vizsgáló szót, amely megállapítja, hogy a két beírt betű betűrendben áll-e!

```
?
?TO MOGOTT? :BETU1 :BETU2
>OP NOT ELOTT? :BETU1 :BETU2
>END
MOGOTT? DEFINED
?
?PR MOGOTT? "A "B
FALSE
?PR MOGOTT? "B "B
TRUE
?PR MOGOTT? "C "A
TRUE
?
```

7. Készítsen olyan vizsgáló szót, amely eldönti, hogy négy bemenet azonos-e!

```
?
?TO MINDAZONOS :A :B :C :D
>OP (AND :A=:B :B=:C :C=:D)
>END
MINDAZONOS DEFINED
?
?PR MINDAZONOS 1 1 1 1
TRUE
?PR MINDAZONOS [illegible]
TRUE
?PR MINDAZONOS [illegible]
FALSE
?
```

2.7.3. Összefoglalás és gyakorlatok

1. A vizsgáló szó a művelet különleges esete. Eredményként "TRUE (igaz) vagy "FALSE (hamis) értéket adnak.
2. A Logo vizsgáló szavakon kívül a felhasználó kívánsága szerint vizsgáló szavakat készíthet.
3. A MEMBERP vizsgáló szó második bemenete kötelezően lista. Ellenkező esetben hibajelzést kapunk.
4. A vizsgáló szavak eredményeit logikai "ÉS"-sel, "VAGY"-gyal és negálással lehet egymáshoz kapcsolni.

Új Logo szavak:

EMPTYP
WORDP
NUMBERP
LISTP
GREATERP
MEMBERP
ASCII
NOT
AND
OR

Feladatok

1. Készítsen olyan vizsgáló szót, amely megállapítja, hogy egy lista utolsó eleme szám-e!
2. Készítsen olyan vizsgáló szót, amely a bemeneti listát megvizsgálja abból a szempontból, hogy a lista utolsó eleme magánhangzó-e!
3. Készítsen olyan vizsgáló szót, amely a pénzfeldobást (fej vagy írás) valósítja meg! Használja erre a célra a RANDOM műveletet! 0 a fej, 1 az írás.
4. Készítsen olyan vizsgáló szót, amely megállapítja, hogy a bemenő szám egész-e!
5. Készítsen olyan vizsgáló szót, amely egy számot megvizsgál hogy az páratlan szám-e! Használja erre a célra a 4. feladatot és a NOT műveletet!
6. Készítsen olyan vizsgáló szót, amely a bemeneti listát megvizsgálja abból a szempontból, hogy az egy szóból, üres listából vagy pedig ötelemű listából áll-e! Használja erre a célra az AND logikai operátort kerek zárójelekkel! Az AND után mindhárom vizsgálatot egymás után fel kell sorolni. Az AND a logikai ÉS-nek felel meg.
7. Készítse el az AND (és az OR) műveletek igazságtábláját!

2.8. Két fontos vezérlő szó

2.8.1. A programelágazás

Az előző fejezetekben szereplő valamennyi program soros végrehajtású volt. Az egyes utasítássorokat a gép szépen egymás után hajtotta végre. Gyakran vannak azonban olyan feltételek, amelyektől valami függ. Képezzünk néhány mondatot „Ha"-val és máris világos lesz az állítás.

„Ha" helyett mondhatjuk azt is, hogy „abban az esetben", „egyébként" helyett, hogy „ellenkező esetben". Angolul a megfelelő szavak a következők:

IF (feltétel),

THEN utasítás az igaz esetre (igen ág),

ELSE utasítás a hamis esetre (nem ág).

Ha csak igaz esetről van szó, akkor egyágú programelágazásról beszélünk, egyébként kétágú programelágazásról.

Az itt alkalmazott Logo változat csak az IF szót használja. A THEN és az ELSE felesleges. A feltétel után vagy egy utasításlistának kell állnia, vagy pedig kettőnek, ha nem ág is van. Az IF után álló feltétel vagy "TRUE vagy "FALSE. Természetesen használhatunk vizsgáló szavakat is. Nézzük a következő példákat: Az időponttól függően a KOSZONES a napszaknak megfelelő köszönést nyomtatja ki:

```
?
TO KOSZONES :NAPSZAK
IF :NAPSZAK<12 [OP [JO ESTET]]
IF :NAPSZAK>12 [OP [JO NAPOT]] [OP [JO
REGGELT]]
END
KOSZONES DEFINED
?
?PR KOSZONES 11.59
JO REGGELT
?PR KOSZONES 14
JO NAPOT
?PR KOSZONES 18.01
JO ESTET
?
```

A következő példában megvizsgáljuk, hogy a beírt betű magánhangzó-e. Eredményként egy kijelentést nyomtat ki a gép:

```
TO EGYMAGANHANGZO? :BETU
IF NOT WORDP :BETU [OP [HIBAS INPUT!]]
IF MEMBERP :BETU [A E I O U] [OP SE :BETU
[EGY MAGANHANGZO!] [OP SE :BETU [NEM MAGANHANGZO!]]]
END

?
?PR EGYMAGANHANGZO? A EN U
HIBAS INPUT!
?PR EGYMAGANHANGZO? "EN
EN NEM MAGANHANGZO!
?PR EGYMAGANHANGZO? "I
I EGY MAGANHANGZO
?
```

Az ELAGAZAS azt mutatja be, hogy nemcsak egyszerű utasításokra ágaz-hatunk el, hanem az elágazásban más programokat is hívhatunk.

```
TO ELAGAZAS :TARGY
IF LISTP :TARGY [PROGRAM1 :TARGY] [PROGRAM2 :TARGY]
PROGRAM3
END

?
TO PROGRAM1 :LISTA
PR [LISTAKAT DOLGOZOK FEL]
PR (SE [MI LESZ A] :LISTA [-VAL?])
END

?
TO PROGRAM2 :SZO
PR (SE [EN FELDOLGOZOM A] :SZO [SZOT])
END

?
TO PROGRAM3
PR [EN VAGYOK AZ ELAGAZAS VEGE]
END

?
?
?ELAGAZAS "HALIHO
EN FELDOLGOZOM A HALIHO SZOT
EN VAGYOK AZ ELAGAZAS VEGE
?ELAGAZAS [A HALIHO]
LISTAKAT DOLGOZOK FEL
MI LESZ A HALIHO -VAL?
EN VAGYOK AZ ELAGAZAS VEGE
?
```

A következő példában egy már ismert, általunk definiált vizsgáló szót használunk és az elágazásba egy újabb elágazást építünk be.

Ennél a zárójeles szerkezetnél már igencsak figyelnünk kell, mivel a hiányzó THEN és ELSE szavak az olvasást megnehezítik. A LISTP: OBJ feltétel után két lista következik. Tehát kétszeres elágazással van dolgunk. A TRUE ágba van beépítve a 3ELEMU vizsgáló szóval végrehajtott művelet.

```
?
?TO ELAGAZAS :OBJ
>IF LISTP :OBJ [IF 3ELEMU :OBJ [PROGRAM1 :OBJ]]
>[PROGRAM2] [PROGRAM3]
>END
ELAGAZAS DEFINED
?
?TO 3ELEMU :LISTA
>OP 3=COUNT :LISTA
>END
3ELEMU DEFINED
?
```

```
?TO PROGRAM1 :LISTA
>PR [ 3 ELEMU LISTAKAT DOLGOZOK FEL ]
>PR (SE [ MI LESZ A ] :LISTA [ -VAL? ])
>END
PROGRAM1 DEFINED
?
TO PROGRAM2
PR [EN VAGYOK A NEM AG]
END

?
?TO PROGRAM3
>PR [ EN VAGYOK AZ ELAGAZAS VEGE ]
>END
PROGRAM3 DEFINED
?
?
?ELAGAZAS "MI?
EN VAGYOK A NEM AG
EN VAGYOK AZ ELAGAZAS VEGE
?
?ELAGAZAS [ HOGYAN, KEREM? ]
EN VAGYOK AZ ELAGAZAS VEGE
?
?ELAGAZAS [ KI VAN ITT? ]
3 ELEMU LISTAKAT DOLGOZOK FEL
MI LESZ A KI VAN ITT? -VAL?
EN VAGYOK AZ ELAGAZAS VEGE
?
```

Az utolsó példában adatellenőrzést végzünk. Csak azok a szavak megengedettek, amelyek mássalhangzóval kezdődnek. Bármilyen hiba, azaz hibás adat esetén hibaüzenet jelenik meg, a program futása az adott ponton befejeződik. Az ADATELLENOR segítségével átismételjük a Logo vizsgáló szavakat. Ha valamelyik feltétel teljesül, (igaz eset), akkor az őt követő OUTPUT kiadja a hibaüzenetet. Az értékek OUTPUT-tal való továbbítása minden esetben azt jelenti, hogy a program befejeződik.

```
TO ADATELLENOR :OBJ
IF LISTP :OBJ [OP [LISTA TILOS]]
IF NUMBERP :OBJ [OP [SZAM TILOS]]
IF EMPTYP :OBJ [OP [URES SZO TILOS]]
IF MEMBERP FIRST :OBJ [A E U I O] [OP [MAGANHANGZOVAL
KEZDODIK]]
RENDBEN [] :OBJ [-VAL]]
END
```

Már megállapítottuk, hogy a kétszeres programelágazások esetében az utasítássor olvashatósága könnyebb is lehetne. A Logo erre is lehetőséget kínál. IF helyett a TEST használható, így is vizsgálható a feltétel. Az eredményt ("TRUE vagy "FALSE) megjegyzi magának a számítógép. Közvetlenül a TEST-et tartalmazó utasítássor után írjuk az igen elágazássort, ez elé kerül

az IFTRUE (IFT). A nem ágat tartalmazó utasítássornak IFFALSE (IFF) utasítással kell kezdődnie. Ez a lehetőség módot ad arra, hogy a „hosszú IF-utasítások" áttekinthetők legyenek. A már ismert EGYMAGANHANGZÓ? példáján lássuk ezt!

```
?
?TO EGYMAGANHANGZO? :BETU
>TEST NOT WORDP :BETU
>IFTRUE Á OP É HIBAS INPUT! Ü
>TEST MEMBERP :BETU Á A E I O U Ü
>IFTRUE Ö OP SE :BETU Ö MAGANHANGZO Ü
>IFFALSE Á OP SE :BETU Á NEM MAGANHANGZO Ü
>END
EGYMAGANHANGZO? DEFINED
?
```

2.8.2. A STOP utasítás

Ezt az utasítást két példa segítségével szemléltetjük. A már ismert KOSZONES-ben OUTPUT helyett tegyünk PRINT utasítást. Mi változik meg? Próbáljuk ki:

```
TO KOSZONES :NAPSZAK
IF :NAPSZAK > 18 ÁPR ÁJO ESTETÜ
IF :NAPSZAK > 12 ÁPR ÁJO NAPOTÜ ÁPR ÁJO REGGELTÜ
END

?
?KOSZONES 19.07
JO ESTET
JO NAPOT
?
```

Az első utasítássor feltétele teljesül. 18.00 már elmúlt, ezért a gép „Jó estét!"-tel köszön. Ezzel tulajdonképpen vége a programnak. Azonban még végrehajtja a következő utasítást és azt mondja, hogy „Jó napot!". Ebben az esetben segít a STOP utasítás. A programozó leállíthatja a megfelelő helyen a programot. A KOSZONES tehát a következőképp alakul:

```
?
?PO "KOSZONES
TO KOSZONES :NAPSZAK
IF :NAPSZAK > 18 ÁPR ÁJO ESTETÜ STOPÜ
IF :NAPSZAK > 12 ÁPR ÁJO NAPOTÜÜ ÁPR ÁJO REGGELTÜÜ
END

?
?KOSZONES 19.07
JO ESTET
?
```

Hasonlóképpen készítsünk egy módosított ADATELLENOR-t! Itt is PRINT utasításokat használtunk az OUTPUT-ok helyett. Mivel az elágazások után nincsenek STOP utasítások, a vizsgáló program egyszer visszabeszél, végül azonban az adatokat mégiscsak jónak találja.

```
TO ADATELLENOR :OBJ
IF LISTP :OBJ [PR [LISTA TILOS]]
IF NUMBERP :OBJ [PR [SZAM TILOS]]
IF EMPTYP :OBJ [PR [URES SZO TILOS]]
IF MEMBERP FIRST :OBJ [A E I O U] [PR [MAGANHANGZOVAL
KEZDODIK]] [PR (SE [MINDEN
RENDBEN A] :OBJ [-VAL])]
END

?
?ADATELLENOR [A EZ EGY LISTA]
LISTA TILOS
MINDEN RENDBEN A EZ EGY LISTA -VAL
?
?ADATELLENOR 9871
SZAM TILOS
MINDEN RENDBEN A 9871 -VAL
?
?ADATELLENOR [A B C]
LISTA TILOS
MAGANHANGZOVAL KEZDODIK
?
```

2.8.3. A programok önmagukat hívják ellenőrzés mellett

```
?
?TO ALATT :OBJ
>IF EMPTYP :OBJ [STOP]
>PR FIRST :OBJ ALATT BF :OBJ
>END
ALATT DEFINED
?
?
?ALATT "TEGNAP
T
E
G
N
A
P
?
?
?ALATT [ITT JARTAM, VARTAM]
ITT
JARTAM,
VARTAM
?
```

Az ALATT tehát egy olyan nyomtató program, amely valamely szó betűit vagy egy lista elemeit egymás alá nyomtatja.

Vizsgáljuk meg a programot soronként! TO után szokás szerint a funkcónév és a funkcióbemenet (:OBJ) következik. A bemenet szó vagy lista lehet.

Az első sor leállási feltétel. Ha az :OBJ programváltozó csak üres szót ('') vagy üres listát ([]) tartalmaz, akkor a program álljon le. A második utasítássor alapján a program kinyomtatja a programváltozó első elemét (PR FIRST:OBJ). Igen, és most az ALATT önmagát hívja. Igen ám, de a bemenet értéke megváltozott, azaz a bemenet a régi bemenet az első betű vagy az első listaelem kivételével (BF:OBJ). Ez egyszerű, de igen hatásos megoldás.

Az ALATT változó bemeneti értékeit szemlélteti a második példa is. A további teszt során írassuk ki az ALATT mindenkori bementét is, így láthatjuk, hogy a BF :OBJ hatására miképpen változik újra és újra az ALATT bemenete.

Második példa a TUKOR. A bemenetként megadott szavakat fordítva írja ki:

```
?
?TO TUKOR :SZO
>IF EMPTYP :SZO [STOP]
>TYPE LAST :SZO TUKOR BL :SZO
>END
TUKOR DEFINED
?
?
?TUKOR "BABSZINHAZ
ZAHNIZSBAB?
?
```

A következő esetekben két bemenet van. NEL a beírt tárgy elejéről előre megadott számú elemet eltávolít. Minden egyes BUTFIRST egy további elemet távolít el. A :SZAM szabja meg, hogy hányszor kell végrehajtani a BUTFIRST-öt, azaz hány elemet kell eltávolítani. A NEL minden egyes újabb hívása során megváltozik a bemenet. Mivel a tárgy első elemét eltávolítottuk, a továbbiak során eggyel kevesebb betűt kell eltávolítani, azaz a hívások száma eggyel csökken (:SZAM–1).

```
?
?TO NEL :OBJ :SZAM
>IF :SZAM=0 [PR :OBJ STOP]
>NEL BF :OBJ :SZAM-1
>END
NEL DEFINED
?
?NEL "KISKOROSI 3
KOROSI
?
?
?NEL [EGY KIS MALAC ROF ROF ROF] 4
ROF ROF
?
```

Vizsgáljuk meg ismét a közbülső futási eredményeket! Megfigyelhető, hogy minden egyes hívás hatására a beadott szó egy betűvel rövidebb lesz és a szám ennek megfelelően eggyel csökken. Ha a szám elérte a nullát, akkor a beírt szó kinyomtatódik.

A NIK program hasonlóan működik, valószínűleg további magyarázatra nincs szükség.

```
TO NIK :OBJ :HELY
IF :HELY = 1 [PR FIRST :OBJ STOP]
NIK BF :OBJ :HELY - 1
END

?
?NIK "VIZILABDA 7
B
?
?NIK [AMI NEM VAGYUNK ITT] 3
VAGYUNK
?
```

A következő program a számjegyeket adja össze. Az összegzésben egy apró trükk van, mert az összeget egy változóban tárolnunk kell. Az új számot mindig hozzáadjuk az eredeti összeg értékéhez. Ezt az :ÖSSZEG változóban tároljuk. A VISSZOSSZEG meghívásakor természetesen :OSSZEG-ben 0-nak kell lennie.

```
?
?
?TO VISSZOSSZEG :SZAM :OSSZEG
>IF EMPTYP :SZAM [PR :OSSZEG STOP]
>VISSZOSSZEG BF :SZAM :OSSZEG + FIRST :SZAM
>END
VISSZOSSZEG DEFINED
?
?VISSZOSSZEG 98723 0
29
?
```

Az NSZER.NYOMTAT hasonlóan működik NEL-hez vagy a NIK-hez. :N számolja a műveleteket. Ha elérte a nullát, akkor a program STOP hatására leáll.

```
TO NSZER.NYOMTAT :MIT :N
IF :N = 0 [STOP]
TYPE :MIT NSZER.NYOMTAT :MIT :N - 1
END

?
?NSZER.NYOMTAT "$ 20
$$$$$$$$$$$$$$$$$$$$?
?
?NSZER.NYOMTAT "HAHIHA 4
HAHIHAHAHIHAHAHIHAHAHIHA?
?
```

Az NSZER.PROG-ban valami újdonság van. Formailag hasonló az NSZER.NYOMTAT-hoz. Híváskor MIT helyére egy programnevet és a hozzá tartozó bemeneteket írjuk. :MIT egy lista kell legyen, amelynek végrehajtására csak akkor kerül sor, amikor a RUN utasítás bemenetévé válik.

```
?
?TO NSZER.PROG :MIT :N
>IF :N=0 [STOP]
>RUN :MIT NSZER.PROG :MIT :N-1
>END
NSZER.PROG DEFINED
?
?NSZER.PROG [NEL [KI VAN ITT ?] 2] 4
ITT ?
ITT ?
ITT ?
ITT ?
?
?NSZER.PROG [TUKOR "SALATA] 4
ATALASATALASATALASATALAS?
?
```

Az NSZER.PROG-ot lényegében már régen ismerjük, hiszen hatása megegyezik a REPEAT utasítással. Ha most az NSZER.PROG helyére ISMETLES-t írunk és felcseréljük a két programváltozót, akkor így is megkaphatjuk a REPEAT utasítás magyar nyelvű változatát.

Befejezésül foglalkozzunk még bizonyítványírással. A tantárgyakhoz tartozó jegyeket ne számként, hanem szövegesen írjuk ki. Használjuk a szokásos osztályzatokat, amelyek jelestől elégtelenig terjednek.

A MINIBIZI programlistában számként megadott osztályzatokat nyomtasson ki. A második sort már elég jól ismerjük. NIK-et már a 2.6.4. szakaszban N.ELEM-ként ismertük meg, de itt NIK nem művelet, hanem tulajdonképpen egy nyomtatási utasítás. A JEGYEK az adatok listáját definiálja.

```
TO MINIBIZI :JEGY
IF EMPTYP :JEGY [STOP]
NIK JEGYEK FIRST :JEGY
MINIBIZI BF :JEGY
END

TO NIK :OBJ :HELY
IF :HELY = 1 [PR FIRST :OBJ STOP]
NIK BF :OBJ :HELY - 1
END

TO JEGYEK
OP [ELEGTELEN ELEGSEGES KOZEPES JO JELES]
END

?
?MINIBIZI [1 5 3 2 4 5]
ELEGTELEN
JELES
```

```
KOZEPES
ELEGSEGES
JO
JELES
?
```

A MINIBIZI-t csak a :TARGYAK bemenettel kellett kibővítenünk és a második sorban a :TARGYAK első elemét kellett kinyomtatni, és máris kész a bizonyítványprogramunk.

```
TO BIZI :JEGY :TARGYAK
IF EMPTYP :JEGY [STOP]
TYPE FIRST :TARGYAK NIK JEGYEK FIRST :JEGY
BIZI BF :JEGY BF :TARGYAK
END

?
?BIZI [5 1 3] [BIOSZ MATEK MAGYAR]
BIOSZJELES
MATEKELEGTELEN
MAGYARKOZEPES
?
```

A próbafuttatásból látható, hogy van egy szépséghiba. Az osztályzatoknak jobbra kellene állniuk, és a kezdőbetűknek egymás alatt kellene lenniük. Rögzítenünk kellene egy olyan pozíciót, amelytől kezdődően minden sorban kinyomtatódik a tantárgy. Ilyen tabulátorfunkció a TAB:

```
?
?TO TAB :HELY
>IF FIRST CURSOR = :HELY [STOP]
>TYPE "ö
>TAB :HELY
>END
TAB DEFINED
?
```

A funkció bemenete a kívánt pozíció, amely 1 és 40 között lehet. A CURSOR Logo funkció a CURSOR pillanatnyi állapotát adja. A CURSOR eredménye kételemű lista. Az első elem a szóban forgó sor pozíciója (oszlop), a második elem a sorszám. TAB mindaddig szóközt nyomtat, míg el nem éri a kívánt nyomtatási pozíciót.

```
?
?PO "BIZI
TO BIZI :JEGY :TARGYAK
IF EMPTYP :JEGY [STOP]
TYPE FIRST :TARGYAK TAB 15 NIK JEGYEK FIRST :JEGY
BIZI BF :JEGY BF :TARGYAK
END
```

```
?
?BIZI [ 5 1 3 ] [ BIOSZ MATEK MAGYAR ]
BIOSZ           JELES
MATEK           ELEGTELEN
MAGYAR          KOZEPES
?
```

2.8.4. Összefoglalás és gyakorlatok

1. A soros működésű programok olyanok, hogy az egyes utasításokon a program sorban végigszalad. Egyetlen utasítássor sem marad ki. A vezérlő szavak segítségével vagy bizonyos programrészekre lehet ugrani, vagy pedig leállási feltételek helyezhetők el.
2. Megkülönböztetünk egyágú és kétágú programleágazásokat. Ennek vezérlő szavai: HA...AKKOR...EGYÉBKÉNT... angolul ez a következőképpen hangzik: IF...THEN...ELSE... IF után mindig egy vizsgáló szónak (feltételnek) kell állnia, amely vagy igaz, vagy hamis (TRUE) (FALSE). Igaz esetben csak a THEN szó után álló utasításokat hajtja végre a gép. Hamis esetben a program az ELSE szó után álló utasításokat hajtja végre. Az általunk használt Logo változatban a THEN és az ELSE elmarad, a két ág egyszerűen szögletes zárójelbe kerül.
3. A jobb áttekinthetőség érdekében a kétágú programelágazásokat a TEST, IFTRUE és IFFALSE Logo szavak segítségével írhatjuk fel.
4. A STOP utasítás a programot tetszőlegesen meghatározott helyen megállítja.
5. Az IF és a STOP utasítással az önmagukat hívó programok jól vezérelhetők. Az ilyen programok futását úgy vezérelhetjük, hogy a bemenet értéke mindig akkor változik, amikor a program újból önmagát hívja. Amennyiben a bemenet üres lista, üres szó vagy nulla, akkor a program működését a STOP megszakítja.
6. A RUN utasítás bemenete egy lista. Ez a lista egy utasítássorozat, amelyeket a RUN végrehajt.

Új Logo szavak:

IF
TEST
IFTRUE (IFT)
IFFALSE (IFF)
STOP
RUN
CURSOR

Feladatok

1. Készítsen olyan műveletet, amely két szám közül eredményül a nagyobbikat adja!
2. Az 1. feladat felhasználásával bővítse a műveletet, úgy, hogy négy szám közül válassza ki a legnagyobbat!
3. Oldja meg az 1. feladatot a TEST, IFTRUE és IFFALSE Logo szavak felhasználásával is!
4. Készítsen olyan műveletet, amely egy pénz feldobását szimulálja és eredménye "FEJ" vagy "IRAS"!
5. Készítsen olyan vizsgáló szót, amely megállapítja, hogy a bemenetként megadott szám pozitív-e! Használja a programelágazás mindkét módját! (IF és TEST)!
6. A 2.4.3. pontban szereplő FORGAS programot módosítsa úgy, hogy a hívások számát bemenetként lehessen megadni!
7. Egy változtatható méretű négyzetet (vagy más, tetszőleges geometriai ábrát) adott számszor forgasson meg!

2.9. Rekurzív műveletek

2.9.1. Önmagukat hívó műveletek

Már megismerkedtünk néhány rekurzív, vagyis önmagát hívó programmal, most rátérünk az önmagukat hívó műveletek tárgyalására. Emlékezzünk vissza az előző fejezetekre. Ott megismerkedtünk a műveletek előnyeivel és alapvető felépítésükkel. Mint tudjuk, az OUTPUT adja tovább az adatokat, kapcsolja össze a műveleteket.

Miképpen lehet egy rekurzív nyomtató programot átírni? Nézzük először a TUKOR programot. Amennyiben rekurzív műveletek, akkor a következőképpen használjuk:

```
?
?PR TUKOR "TEGNAP

PANGETTUKOR DIDN'T OUTPUT TO PRINT
```

Nyilvánvaló, hogy nem működik. Elvileg már tudjuk is, hogy nyomtatási utasítások nem szerepelhetnek, de az OUTPUT feltétlen ott kell, hogy legyen.

```
TO TUKOR :SZO
IF EMPTYP :SZO [OUTPUT " ]
OUTPUT WORD LAST :SZO TUKOR BL :SZO
END
```

Vizsgáljuk meg a rekurzió magját, a második utasítássort! Eltűnt a TYPE nyomtatási utasítás, ellenben ott van az OUTPUT. A BL :SZO bemenettel való önhívás nem meglepő. De mit keres itt a WORD? Keressünk erre magyarázatot, mivel a rekurziót a későbbiekben még sokkal behatóbban fogjuk vizsgálni!

A régi TUKOR minden lépésben pontosan egy betűt – nevezetesen az utolsó betűt – nyomtatta ki. A bemenetből elmaradt az utolsó betű, majd az egész megismétlődött.

Az új TUKOR az utolsó betűt nem felejtheti egyszerűen el, hanem az összeredmény részeként tárolja. Így az egyes betűket a WORD segítségével szólánccá illesztjük össze. Nyomtatás helyett a program a jeleket egymás mellé állítja. A leállás feltétele, hogy az OUTPUT üres szót talál.

Most már működik a PR TUKOR utasítás, és műveletek egymásutánja is lehetséges.

```
?
?PR TUKOR "TEGNAP
PANGET
?
?PR TUKOR TUKOR "TEGNAP
 TEGNAP
?
```

A rekurziók listákkal is működnek. Ha a TUKOR-nek egy lista elemeit kell megfordítania, akkor csak a listákra vonatkozó megfelelő Logo funkciókat kell beépítenünk. Nevezzük ezt a módosított változatot TUKORL-nek. Az utolsó L betű a listaváltozatra utal.

```
?TO TUKORL :MONDAT
>IF EMPTYP :MONDAT [OP []]
>OP SE LAST :MONDAT TUKORL BL :MONDAT
>END
TUKORL DEFINED
?
?
?PR TUKORL [EN TE O MI TI]
[I MI O TE EN]
?
```

A következő példában TUKORL-t tovább módosítjuk úgy, hogy ne csak a listaelemeket, hanem a szavakat is fordítsa meg. Lássuk ezt a műveletet a példával együtt:

```
TO TUKORL :MONDAT
IF EMPTYP :MONDAT [OP []]
OP SE TUKOR LAST :MONDAT TUKORL BL :MONDAT
END

?
?PR TUKORL [É EN NEM VAGYOK ITT! ]
!TTI  KOYGAV  MEN  NE
?
```

A régi NEL-t is bemutatjuk rekurzív műveletként:

```
TO NEL :OBJ :SZAM
IF :SZAM = 0 [OUTPUT :OBJ]
OUTPUT NEL BF :OBJ :SZAM - 1
END

?
?PR NEL "KISKOROS 3
KOROS
?
```

Most definiáljunk új rekurzív műveleteket! Az első művelet egyetlen jelből képez jelláncot. A kívánt jelet és a láncban szereplő betűk számára programváltozóként adtuk meg.

```
?
?TO TOLTO :JEL :SZAM
>IF :SZAM=0 [OP "o]
>OP WORD :JEL TOLTO :JEL :SZAM-1
>END
TOLTO DEFINED
?
?
?PR TOLTO "$ 15
$$$$$$$$$$$$$$$
?
?PR TOLTO "_ 20

--------------------
?
?PR TOLTO "% 15
%%%%%%%%%%%%%%%
?
```

A SZOBOLLISTA művelet szavakból képez listákat. A lista egyes elemei a szó betűi:

```
?
?TO SZOBOLLISTA :SZO
>IF EMPTYP :SZO [OP []]
>OP SE FIRST :SZO SZOBOLLISTA BF :SZO
>END
SZOBOLLISTA DEFINED
?
?
?PR SZOBOLLISTA "RIGO
R I G O
?
?PR SZOBOLLISTA "TEGNAP
T E G N A P
?
?PR COUNT SZOBOLLISTA "TEGNAP
6
?
?PR MEMBERP "G SZOBOLLISTA "TEGNAP
TRUE
```

```
?PR MEMBERP "X SZOBOLLISTA "TEGMAF
FALSE
?
```

A SZOBOLLISTA rekurziót felhasználhatjuk arra, hogy két új hasznos műveletet képezzünk. A COUNTW meghatározza a szóban szereplő betűk számát, MEMBERPW megvizsgálja, hogy a bemenetként megadott betű szerepel-e a betűláncban.

```
?
?TO COUNTW :SZO
>OP COUNT SZOBOLLISTA :SZO
>END
COUNTW DEFINED
?
?
?TO MEMBERPW :JEL :SZO
>OP MEMBERP :JEL SZOBOLLISTA :SZO
>END
MEMBERPW DEFINED
?
?
?PR COUNTW "ANNAMARIA
9
?
?PR MEMBERPW "R "ANNAMARIA
TRUE
?
?PR COUNTW TOLTO "$ 15
16
?
?PR COUNTW TOLTO "% 20
21
?
?PR TOLTO "A COUNTW "ANNAMARIA
AAAAAAAAA
?
```

További példák a gyakorlatok között találhatók.

2.9.2. Összefoglalás és gyakorlatok

1. A rekurziók olyan programok, amelyek önmagukat hívják. A rekurzív műveletek a felhasználó által előállított olyan funkciók, amelyek feldolgozásuk eredményét OUTPUT-tal adják tovább.
2. Minden rekurzív művelet lényegében három részre osztható:
   - befejezési feltétel;
   - feldolgozó rész;
   - ismételt programhívás módosított bemenettel.

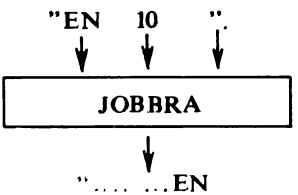
3. A leállási feltétel elérésekor a művelet úgy fejeződik be, hogy OUTPUT-tal egy értéket ad tovább. Ez szavakkal vagy listákkal végzett műveletek esetében gyakran az üres szó vagy az üres lista.
4. A feldolgozó rész kijelöli, hogy mit kell tenni akkor, ha a befejezési feltétel nem teljesül. Ez a rész az ismételt felhívással együtt adja a rekurzió magját. Szavakkal vagy listákkal végrehajtott művelet során közvetlenül az OUTPUT szó után álló elemmel hatja végre a gép az utasítást.
5. Az ismételt programfelhívás bemenete eltér az előzőétől. Szavak vagy listák esetében az első elemet általában BUTFIRST-tel távolítjuk el. A rekurziót vezérlő számok esetében általában 1-et vonnak le.
6. Annak érdekében, hogy a listákra megírt Logo funkciók szavakra is alkalmazhatók legyenek, olyan művelet készíthető, amely a szót listává alakítja. Minden egyes szó ily módon a lista elemévé válik. Erre a listára már alkalmazható a megfelelő Logo funkció. A rekurzív műveletek előnye a rekurzív utasításokkal szemben az, hogy más funkciókkal láncolhatók.

Új Logo szavak:

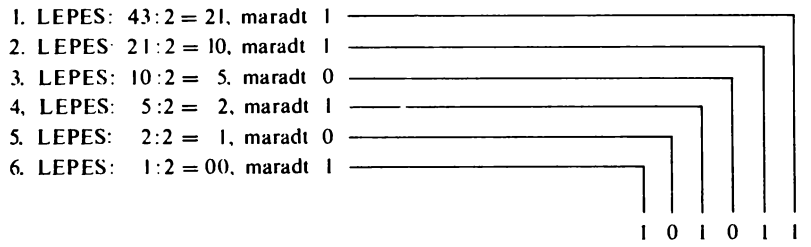
SUM QUOTIENT (a feladatok kapcsán) FPUT (a feladatok kapcsán).

Feladatok

1. Képezzen olyan rekurzív műveletet, amely egy lista elemeit szóvá (összefüggő jellánccá) alakítja! A lista minden eleme betű vagy szó legyen!
2. Állítson elő olyan műveletet, amely meghatározza egy szó vagy egy lista előre megadott elemét!
3. A JOBBRA művelet olyan legyen, hogy a bemenetként megadott szó elé tetszőlegesen választott karaktereket ír. Mind a karakter, mind a szó bemenet. Használja erre a célra a már ismert TOLTO funkciót!



4. A MAGHANGGYILKOS művelet a bemenetként megadott szó magánhangzóit ponttal helyettesíti. Állítsa elő!
5. Alakítsa át a tizedes számokat maradékos osztási eljárással bináris számokká! A maradékos osztási eljárást vázlatosan a 43 decimális szám átalakításának példáján mutatjuk be:



6. Készítsen olyan vizsgáló szót, amely az első szót összehasonlítja a második szóval, és megállapítja, hogy betűrendben állnak-e!
7. Készítsen olyan vizsgáló szót, amely megvizsgálja, hogy a listában vannak-e üres listaelemek! Amennyiben talál üres elemet, akkor "TRUE, ellenkező esetben "FALSE eredményt adjon!
8. Legyen a művelet bemenete egy tetszőleges szó és egy betűrendbe állított mondat (lista). A szót a mondatba betűrend szerint kell beilleszteni!
9. Készítsen olyan műveletet, amely valamely lista szavait betűrendbe állítja! Ehhez felhasználhatók a 8. és 6. feladatok eredményei.

2.10. Ugrás és végrehajtás

2.10.1. Ugrási utasítás

A GO utasítás lehetővé teszi, hogy a programon belül több utasítássort átugorjunk. A beérkezés helyét külön jelölnünk kell. Lássunk egy egyszerű példát, amely egyfolytában, végtelen hurokban nyomtatja a ~~"HUROK szót~~ A program végéről ismét a program elejére ugrunk.

```
TO HUROK
LABEL "KEZDET
PR [INDUL A GOROG ALUDNI]
GO "KEZDET
END
```

```
"HUROK
INDUL A GOROG ALUDNI
INDUL A GOROG ALUDNI
INDUL A GOROG ALUDNI
INDUL A GOROG ALUDNI
INDUL A GOROG ALUDNI
INDUL A GOROG ALUDNI
INDUL A GOROG ALUDNI
INDUL A GOROG ALUDNI
INDUL A GOROG ALUDNI
INDUL STOPPED! IN HUROK:
PR AINDUL A GOROG ALUDNIU
?
```

A LABEL-lel, amelyet egy szó követ, a programban jelölhetjük a beugrás helyét. A GO utasítás bemenete a beugrási hely neve. A következő szakaszokban rengeteg olyan példa lesz, amely többek között ilyen ugrási utasításokat használ.

2.10.2. Értékadás címmel és tartalommal

Eddig az adattárolókat a funkcióhívások során töltöttük fel értékekkel. Ismerkedjünk most meg azokkal a lehetőségekkel, amelyeket felhasználva egy funkción belül bizonyos objektumokat névvel láthatunk el, és ezzel a névvel ezekre az objektumokra hivatkozhatunk. Az adattárolókat szemléletesen fióknak tekinthetjük, természetesen sok ilyen fiók van. A tárgyak könnyű fellelése érdekében minden fiókra láthatóan rá van írva egy név. Ha kihúzzuk a fiókot, akkor meglátjuk a tartalmát.

A "HONAP nevű fiók, pl. a "JUNIUS szót tartalmazza. A "GEPKOCSITIPUSOK fiók pl. egy öt szóból álló listát: [BVW VW OPEL LADA FORD]. A Logoban az adattárolás a MAKE utasítással definiálható. A MAKE-nek két bemenete van. Ezt értékadásnak nevezzük. A "HONAP változó a "JUNIUS szót kapja értékül. A "GEPKOCSITIPUSOK változó értékadáskor ötelemű listát kap.

```
?
?MAKE "HONAP "JUNIUS
?MAKE "GEPKOCSITIPUS [BMW WV OPEL LADA FORD]
?
```

A változók tartalmát a következőképpen hívhatjuk le:

```
?
?PR THING "HONAP
JUNIUS
?PR THING "GEPKOCSITIPUS
BMW WV OPEL LADA FORD
?
```

```
?
?PR :HONAP
JUNIUS
?PR :GEPKOCSITIPUS
BMW WV OPEL LADA FORD
?
```

A kettőspontos írásmód egyszerűbb és megszokottabb.

Lássunk egy kis példát. Egy függvény a "LOTTOTIP változóba írjon be 5-től 90-ig terjedő számot.

```
?
?TO LOTTOSZAMOLO
>MAKE "LOTTOTIP []
>LABEL "KEZDET
>MAKE "SZAM SUM 1 RANDOM 90
>TEST MEMBERP :SZAM :LOTTOTIP
>IFF [MAKE "LOTTOTIP SE :LOTTOTIP :SZAM]
>IF COUNT :LOTTOTIP=5 [STOP]
>GO "KEZDET
>END
LOTTOSZAMOLO DEFINED
?
```

Ha most PONS Printout Names felhasználásával a tárolóban levő összes változót kiíratjuk, akkor nemcsak a jelenlegi, hanem az előző változók is felkerülnek a képernyőre.

```
?
?LOTTOSZAMOLO
?PONS
LOTTOTIP IS [82 8 54 67 42]
GEPKOCSITIPUS IS [BMW WV OPEL LADA FORD]
HONAP IS JUNIUS
SZAM IS 42
?
```

A NAMEP viszgáló szó felhasználásával megállapíthatjuk, hogy egy változónevet már használtunk-e.

```
?
?PR NAMEP "LOTTOSZAMOLO
FALSE
?
?PR NAMEP "LOTTOTIP
TRUE
?
```

A fölösleges változókat az ERN (Erase Names) törlési utasítással tüntethetjük el, megadva a változó nevét.

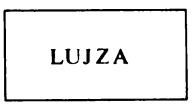
```
?
?ERN [HONAP GEPKOCSITIPUS]
?PONS
LOTTOTIP IS [82 8 54 67 42]
SZAM IS 42
?
```

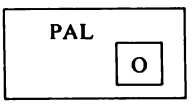
Befejezésül lássunk egy olyan lehetőséget, amelyet sok programozási nyelv nem eneged meg. Változókban más változók nevét tárolhatjuk, és ezen változók tartalmát kinyomtathatjuk anélkül, hogy a nevüket ismernénk. Szemléltetésként először két változót definiálunk, majd ezek tartalmát kilistázzuk a PONS segítségével.

```
?MAKE "EN "PAL
?MAKE "TE "LUJZA
?
?PONS
TE IS LUJZA
EN IS PAL
?
```

(Magyarul olvasva kissé mulatságos lett.)

A következő értékadás során az EN (én!) változónak az "O (ő!) értéket adjuk. Ez csak úgy lehetséges, ha EN egy szó változónak a neve.





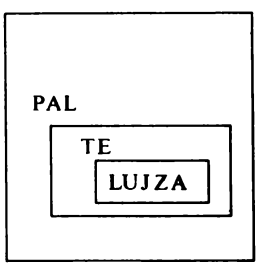

```
?
?MAKE :EN "O
?PONS
TE IS LUJZA
PAL IS O
EN IS PAL
?
```

Ha ugyanis EN értékéül a 0 értékét adjuk, akkor O-nak nincs további értéke, ahogy a Logo is közli.

```
?
?MAKE :EN :O
O HAS NO VALUE
?
```

Ha az EN változóba a TE változót tesszük, akkor már egymásba skatulyázottakká válnak.

```
?MAKE :EN :TE
?PONS
TE IS LUJZA
PAL IS LUJZA
EN IS PAL
?
?PR :EN
PAL
?PR THING :EN
LUJZA
?
```



2.10.3. Programciklus

Többszörös ismétlések úgyis megvalósíthatók, hogy megfelelően vezérelve a program elejére ugrunk. A rekurzív funkciókkal ellentétben ezeket iteratív funkcióknak nevezzük. Elvileg minden ismétlés iteratív vagy rekurzív úton egyaránt megfogalmazható. Ebben a szakaszban a már ismert példákat iteratív úton fogalmazzuk meg.

Lássuk a régről ismert TUKOR példát:

```
TO TUKOR :OBJ
MAKE "EREDMENY "
LABEL "START
IF EMPTYP :OBJ [OP :EREDMENY]
MAKE "EREDMENY WORD :EREDMENY LAST :OBJ
MAKE "OBJ BL :OBJ
GO "START
END

?
```

Ez a program most művelet jellegű, mivel az eredményt OUTPUT-tal adja tovább. A feldolgozó rész a "START belépési pont és a GO utasítás között van. Minden egyes végrehajtás során veszi az OBJ utolsó betűjét és az :EREDMENY tároló tartalmának végére illeszti. A leállási feltétel közvetlenül a belépési pont után van. Fontos, hogy az :EREDMENY változót deklarálni kell. Ellenkező esetben a program futása megszakad és megjelenik az EREDMENY HAS NO VALUE hibaüzenet. A WORD :EREDMENY LAST :OBJ utasítás nem hajtható végre, mivel az EREDMENY változót nem hoztuk létre, és így nem is lehet tartalma.

NEL egy ún. számláló ciklus. A bemenetként megadott számszor kerül sor a belépési pont és a GO utasítás közötti utasítások végrehajtására.

```
TO NEL :OBJ :SZAM
LABEL "START
IF :SZAM = 0 [OP :OBJ]
MAKE "OBJ BF :OBJ
MAKE "SZAM :SZAM - 1
GO "START
END
```

Az ilyen számlálóhurkok helyett sokkal kényelmesebb a REPEAT utasítást használni. Ezt a lehetőséget szemlélteti a NEL1.

```
TO NEL1 :OBJ :SZAM
REPEAT :SZAM [MAKE "OBJ BF :OBJ]
OP :OBJ
END
```

Próbaképpen ellenőrizzük, hogy a munkatárban milyen változók vanak:

```
?PONS
EREDMENY IS  PANGET
```

Csak egy változót találunk, annak ellenére, hogy az előző példákban más változókat is használtunk. Azok a változók ugyanis, amelyek a funkciók fejlécében definiálva vannak, nem szerepelnek a munkatárban. Erre a sajátosságra még vissza fogunk térni.

2.10.4. Összefoglalás és gyakorlatok

1. A Logo további vezérlő lehetősége a GO ugrási utasítás. A GO-t a LABEL-lel együtt használják. A LABEL jelöli a belépési pontot. A GO és LABEL bemenetének azonosnak kell lennie. A GO utasítással a programból nem lehet kilépni.
2. Az adattáraknak nevük és tartalmuk van. A változó neve mindig Logo szó. A tartalmuk szavak, számok vagy listák lehetnek.
3. Változó MAKE utasítással állíthatók elő. A MAKE-nek két bemenete van. Az első a változó neve, a második a tár tartalma.
4. A tár tartalma lehívható, ha a változó neve elé a THING műveletet írjuk. A THING rövidítése kettőspont, amelyet a változónév idézőjelének helyére kell írni.
5. Ciklikus műveletek ugyancsak könnyen elvégezhetők, ha GO utasítással a program elejére ugrunk.
6. Mielőtt a programok változóit feldolgoznánk, definiálnunk kell őket. A program elején célszerű minden felhasználandó változót felsorolni, értékül üres szót vagy üres listát kell beírni. A funkció fejlécében definiált változókat természetesen nem kell deklarálni.
7. Az egyszerű számoló hurkokat célszerű "REPEAT utasítással megoldani.

Új Logo szavak:

GO
LABEL
MAKE
THING
ERN
PONS
NAMEP (a gyakorlatok kapcsán)

Feladatok

1. Az előzőkben bemutatott példákat követve írja át a 2.9. fejezet 1., 2., 4. és 5. feladatát iterációvá!
2. Készítsen olyan „ugrólécet", amely a beírt számtól függően megfelelő programrészekre ágaztat el!
3. Készítsen olyan szövegelemző programot, amely egy jelsorozatot abból a szempontból vizsgál meg, hogy hány jel fordul elő benne! Áttekinthető táblázatként betűrendben minden betű mellé a felhasználások számát nyomtassa ki!

3. Logo haladóknak

3.1. Lokális és globális változók

A 10. fejezetben megismerkedtünk azzal a lehetőséggel, hogy változókat hozzunk létre úgy, hogy egy-egy szóhoz bizonyos tartalmat rendelünk. A MAKE utasítással egy felhasználói funkción belül új tárhelyeket hoztunk létre. A funkcióbemenetek nevéhez a felhasználói funkción belül ugyancsak rendelhetünk új értéket. Ezek a funkció fejlécében felsorolt nevek a funkció hívásakor aktuális értéket kapnak. Ezek a paraméterek csak a felhasználói funkción belül ismertek, tehát kizárólag ez a funkció használhatja őket, természetesen a funkció további hívásaikor is. Ezeket a funkcióparamétereket elszigetelt vagy lokális paramétereknek, ill. változóknak nevezzük. A funkcióbemeneteknek ez a lokális jellege teszi lehetővé, hogy egymás után azonos paraméternévvel rendelkező funkciókat definiálhassunk anélkül, hogy ebből gondok származnának.

Tekintsük most az egy szám számjegyeit összeadó VISSZOSSZEG program egy új változatát, amely már az előző fejezetben tárgyalt MAKE értékadást használja, és egy ciklussal oldja meg a feladatot. A VISSZOSSZEG-ben két változónév van: "SZAM és "OSSZEG. A "SZAM egy funkcióparaméter, amely VISSZOSSZEG hívásakor aktuális értéket kap. Az OSSZEG változót a beírt szám egyes számjegyeinek összegzésére használjuk. A program futásának befejezése után a munkatárban csak az "OSSZEG változó található. "SZAM lokális változó volt, így elfelejtődött. A "SZAM változóval ellentétben az "OSSZEG változó globálisnak tekintendő. Az ilyen globális változókhoz bármikor bármelyik felhasználói funkció hozzáférhet. Közvetlen utasításként a PONS alkalmas arra, hogy a munkatár valamennyi globális változóját kiírassa. A programokon belül a PONS csak a funkció számára ismert változókat írja ki.

Ha a véletlenül keletkezett globális változók feleslegesen hevernek a munkatárban, akkor váratlan és megmagyarázhatatlan jelenségeket idézhetnek elő, amikor pl. ugyanezzel a változónévvel hajtunk végre műveleteket. Tanácsos

ezért a LOCAL utasítással a munkatár érvényességét rögtön az adott funkció érvényességi tartományára korlátozni. Ráadásul így nem kötünk le feleslegesen tárterületet. Lássuk tehát a VISSZOSSZEG felhasználói funkciót, és egyúttal ellenőrizzük a működését is:

```
?
TO VISSZOSSZEG :SZAM
LOCAL "OSSZEG
MAKE "OSSZEG 0
LABEL "KEZDET
IF EMPTYP :SZAM [OP :OSSZEG]
MAKE "OSSZEG :OSSZEG + FIRST :SZAM
MAKE "SZAM BF :SZAM
GO "KEZDET
END

?
?PONS
?
?PR VISSZOSSZEG 123
6
?
```

A következőkben azzal a kérdéssel foglalkozunk, hogy a funkciók miképpen reagálnak az azonos nevű lokális és globális változókra.

A DEM01-ben a "NEV lokális változót használjuk. A globális változó szintén "NEV, amelynek tartalma legyen "PALI. A programban is és a program futása után is kiadjuk a PONS utasítást, így kinyomtatjuk a változókat. Hívjuk a DEM01 programot és aktuális értékként a globális változó tartalmát írjuk be!

```
?
TO DEM01 :NEV
MAKE "NEV FIRST :NEV
PONS
END

?
?PONS
NEV IS PALI
?
?DEM01 :NEV
NEV IS P
?
?PONS
NEV IS PALI
?
```

Láthatjuk, hogy a globális változóval azonos nevű lokális változón végzett műveletek a globális változó tartalmát nem változtatják.

DEM02 is az előző kijelentést igazolja:

```
?
?
?TO DEMO2 :NEV
>PR :NEV
>MAKE "NEV WORD :NEV :NEV
>PR :NEV
>END
DEMO2 DEFINED
?
?
?PONS
NEV IS PALI
?
?DEMO2 :NEV
PALI
PALIPALI
?
?PONS
NEV IS PALI
?
?
```

A DEM03-ban a lokális változó neve eltérő. Hajtsunk most végre műveleteket a "NEV változóval, ez a globális változóra is hat:

```
?
?
?TO DEMO3 :SZO
>PR :SZO
>MAKE "NEV BF BF BF :NEV
>PONS
>END
DEMO3 DEFINED
?
?
?PONS
NEV IS PALI
?
?DEMO3 "NEV
NEV
SZO IS NEV
NEV IS I
?
?PONS
NEV IS I
?
?
```

```
?
?PONS
NEV IS PALI
?
?DEMO3 :NEV
PALI
SZO IS PALI
NEV IS I
?
?PONS
NEV IS I
?
```

A DEM04-ből kitűnik, hogy a DEM03-ban tapasztalt jelenség a globális változó nevének ismerete nélkül is bekövetkezhet.

```
?TO DEMO4 :SZO
>PR :SZO
>MAKE :SZO BF BF BF THING :SZO
>PONS
>END
DEMO4 DEFINED
?
?
?PONS
NEV IS PALI
?
?DEMO4 "NEV
NEV
SZO IS NEV
NEV IS I
?
?PONS
NEV IS I
?
?
```

Befejezésül megállapítjuk, hogy globális változókat csak akkor tanácsos használni, ha tényleg globálisan szükség van rájuk. Egy másik példa lesz a függvényértékek sorozata a függvényeket ábrázoló programban, ahol a vezérlő műveleteknél és a grafikus kiíratáskor is fel kell használni az új értékeket.

A funkcióparaméterek nevei mindig lokálisak. Azokat a segédértékeket, amelyek csak egy funkcióra vagy ennek alárendelt funkcióira korlátozódnak, LOCAL segítségével mindig lokális értékként kell definiálni. A lokális változók alkalmazásával elkerüljük azt is, hogy két globális változó véletlenül azonos nevet kapjon, és így azzal a funkcióval, amely ezt a változót másodszor használja, a globális változó eredeti értékét véletlenül töröljük (pl. egy új értékkel átírjuk).

3.2. A munkatár kezelése

3.2.1. Csoportos törlés, írás és elrejtés

Igen gyakran előfordul, hogy a munkatárban egy sereg felhasználó által definiált funkció van, és mágneslemezről további funkciókat akarunk még beolvasni. Ez általában olyankor következik be, ha a programfejlesztés során előző programok funkcióit is felhasználjuk és inkább betöltjük, hogy ne kelljen újra bepötyögni őket.

Hogyan tudunk elegánsan és kényelmesen megoldani ilyen feladatokat? Ismerkedjünk meg a PACKAGE, PKGALL, BURY és UNBURY utasításokkal!

A funkciók ER és ERPS utasításokkal való törlését már ismerjük. ERPS a munkatár valamennyi funkcióját, míg ER csak a megnevezett funkciót vagy pedig a listában megadott funkciókat törli.

Ha pl. a tárban 5 felhasználói funkció van, és ezek közül csak kettőt szeretnénk lemezen tárolni, akkor az eddig alkalmazott módszer hosszadalmas. A két kívánt funkciót azonban külön jellel lehet ellátni, hogy a törlés ellen védve legyenek és külön tárolhatókká váljanak.

Tegyük fel, hogy az A,B, C,D, E nevű programokból csak A és C kell. Ezt PACKAGE segítségével megőrizzük.

```
?
?
?POTS
TO E
TO D
TO C :C1 :C2
TO A :A1
TO B :B1
?
?
?PACKAGE "KELL [A C]
?
```

A PPS (Print out Properties) kinyomtatja az egyes szavaknak adott tulajdonságokat. Jelen esetben az elraktározott A és C funkciót.

Az A és C funkciónevek PROCPKG (Procedure Package) tulajdonsággal rendelkeznek, az értékük "KELL. A PLIST (Property List) utasítással is célirányosan lekérdezhetjük a funkciónevek tulajdonságait:

```
?
?PPS
C'S PROCPKG IS KELL
A'S PROCPKG IS KELL
AIDS'S BURY IS TRUE
.SYSTEM'S BURY IS TRUE
?
?
?PRINT PLIST "A
PROCPKG KELL
?
?
?PRINT PLIST "C
PROCPKG KELL
?
?
```

Az ilyen csomagokat törlés ellen úgy védhetjük, hogy leírjuk a BURY utasítást, majd a csomag nevét. Ezután a munkatárból egyetlen utasítással minden funkciót eltávolíthatunk, és az elrejtett funkciókat megint előhozhatjuk, sőt mágneslemezre írhatjuk.*

```
?
?
?
?BURY "KELL
?ERPS
?POTS
?
?
?
?PPS
KELL'S BURY IS TRUE
AIDS'S BURY IS TRUE
.SYSTEM'S BURY IS TRUE
?
?
```

Amennyiben egy ilyen csomagot mágneslemezre írunk, akkor minden egyes funkciójának tulajdonságai is megőrződnek. Ily módon a munkatárba olyan csomagok is beolvastathatók, amelyek kizárólag rejtett funkciókból állnak. Ha a listákat a POTS vagy POALL segítségével megjelenítjük, a rejtett funkciók nem írják tele a képernyőt.

```
?
?SAVE "AC1
2 PROCEDURES SAVED
?
?SAVE "KELL
2 PROCEDURES SAVED
?
?
```

A kicsomagolást az UNBURY utasítással végezhetjük el.

```
?UNBURY "KELL
?
?POTS
TO C :C1 :C2
TO A :A1
?
?
```

Bemutatjuk még a PKGALL hatását. Ezzel mindent – beleértve a globális változókat is – egy csomaggá tudunk összekötni és ezt a csomagot tulajdonságokkal is elláthatjuk.

* A PPS utasítás a Logo programlemez minden funkcióját kiírja, így az AIDS és SYSTEM funkciót is.

Figyeljük meg, hogy a "DOBOZ változóként használt szó "CSOMAG értékű VALPKG (Value Package) tulajdonságra tett szert.

Amennyiben BURY-val elrejtjük a "CSOMAG-ot, akkor ez és a :DOBOZ globális változó láthatatlanná válik. PONS és ERNS erre a változóra nincsenek hatással. Amennyiben a felhasználói funkciók más-más csomgokban vannak, akkor a globális változókat elkülönítve tárolhatjuk a lemezen.

```
?
?MAKE "DOBOZ "CSOKI
?PKGALL. "CSOMAG
?
?PPS
DOBOZ'S VALPKG IS CSOMAG
C'S PROCPKG IS KELL
A'S PROCPKG IS KELL
AIDS'S BURY IS TRUE
.SYSTEM'S BURY IS TRUE
?
?
?PR THING "DOBOZ
CSOKI
?
?PR PLIST "DOBOZ
VALPKG CSOMAG
?
```

A csomaggá összefogott változók és funkciók a csomagnév felhasználásával

törölhetők:	ERPS	<csomagnév>
elrejthetők:	BURY	<csomagnév>
láthatóvá tehetők:	UNBURY	<csomagnév>
kimenthetők:	SAVE	<fájlnév>
		<csomagnév>

Próbálgassuk tovább ezeket a funkciókat, hogy valóban megismerkedjünk velük. Megjegyezzük, hogy sok funkciónév esetén a csomagtulajdonságok is igényelnek munkatárat.

Az egyes tulajdonságokat a REMPROP (Remove Property) utasítással lehet megszüntetni.

```
?
?
?PPS
DOBOZ'S VALPKG IS CSOMAG
C'S PROCPKG IS KELL
A'S PROCPKG IS KELL
AIDS'S BURY IS TRUE
.SYSTEM'S BURY IS TRUE
?
?
?REMPROP "C "PROCPKG
?PPS
```

```
DOBOZ'S VALPKG IS CSOMAG
A'S PROCPKG IS KELL
AIDS'S BURY IS TRUE
.SYSTEM'S BURY IS TRUE
?
?
?
?
?
?PPROP "C "PROCPKG "KELL
?PPS
DOBOZ'S VALPKG IS CSOMAG
C'S PROCPKG IS KELL
A'S PROCPKG IS KELL
AIDS'S BURY IS TRUE
.SYSTEM'S BURY IS TRUE
?
?
```

Amennyiben felhasználói funkcióknak több, mint 10 ilyen csomagtulajdonságát kell megszüntetni, akkor már segítséget nyújt a következő segédprogram, amely a listában megadott nevű funkciók tulajdonságait törli.

```
?
?
?
?TO REMPROPL :L
>IF EMPTYP :L [STOP]
>REMPROP FIRST :L "PROCPKG
>REMPROPL BF :L
>END
REMPROPL DEFINED
?
?
?REMPROPL [A C]
?
?PPS
DOBOZ'S VALPKG IS CSOMAG
AIDS'S BURY IS TRUE
.SYSTEM'S BURY IS TRUE
?
?
```

3.2.2. Ha kicsi a munkatár

Minden számítógép munkatára korlátozott. Valamikor ki kell belőle a tartalmának egy részét vinni, hogy helyet teremtsünk más programok és adatok számára. Különösen akkor kell helyet csinálni, ha nagyobb programokat írunk, vagy a mágneslemezről programrészeket töltünk be.

Aki sokat dolgozik rekurziókkal, az bizonyára találkozott már a bosszantó üzenettel, miszerint már nincs hely (OUT OF SPACE). Még ha a munkatár

szinte üres is, programhibák könnyen megtölthetik. Bizonyítja ezt a következő példa is:

```
?
?TO TOLTO :JEL :SZAM
>IF :SZAM = 0 [OP "]
>OP WORD :JEL TOLTO :JEL :SZAM-1
>END
TOLTO DEFINED

?
?PR TOLTO "X -2
OUT OF SPACE:
 JUST BEFORE LEAVING TOLTO
?
```

A rekurzív program minden új híváskor egy megfelelő segédtárat vesz igénybe a bemeneti és kimeneti értékek közbülső tárolására. A TOLTO rekurzióban a negatív bemeneti értékekre (–5, –6, –7, ...), a leállási feltétel sosem teljesül. A számítógép újabb és újabb közbülső tárakat nyit, a TOLTO újra és újra hívja a TOLTO-t, végül a számítógép összeomlik, mivel már nincs több munkatár.

A programozás során szívleljük meg a következő szabályokat:

1. Jó módszer, ha a lehető legritkábban használunk globális változókat. Így nem kell arra ügyelnünk, hogy a globális változók ellenőrizetlenül szaporodnak, észrevétlenül a tárban rejtőzködnek és feleslegesen igénybe veszik a helyet. A feleslegessé vált értékeket törölni kell.
2. Nagy programok esetében, amelyek használatakor egyidejűleg igen sok funkciót kell a tárban tartani, a tárigényes futási szakaszokban általában csak az segít, hogy a rekurzív funkciókat iteratív funkciókká írjuk át. Az iterációk futási ideje is kisebb.
3. Ha a változókat vagy függvényeket csomagokká kapcsoljuk össze, akkor ez is tárolóhelyet igényel. Ha nehézségeink vannak, akkor PPS-sel a csomagokat vizsgáljuk meg, és szabadítsunk fel helyet! Különösen ügyelni kell a rejtett funkciókra és változókra!
4. Ha a programok túlságosan megnőnek, akkor részprogramokká kell bontani őket. A mágneslemezről mindig a szükséges részt töltjük be. A feleslegessé vált programokat töröljük, és így helyet teremtünk a betöltendő programok számára.

Helytelen programozási módszer a programneveket és változóneveket egy-két betűre korlátozni. Ezzel csak néhány száz jelnyi munkatárat takarítunk meg.

3.3. Képernyő- és kurzorvezérlés

Az előző fejezetekben már megismerkedtünk a grafikai és a szöveges képernyő törlésére alkalmas utasításokkal. Grafikus üzemmódban bemutattuk azt a lehetőséget, hogy akár a teljes képernyőt, akár az osztott képernyőt hogyan használjuk. A grafikai üzemmódból szöveges üzemmódba a TEXTSCREEN utasítással tudunk átkapcsolni. A szöveges üzemmódból grafikai üzemmódba úgy kapcsolhatunk át, hogy valamilyen grafikai utasítást írunk be. A Logo rendszer ekkor az osztott képernyős lehetőséget választja (SPLITSCREEN) Foglaljuk össze még egyszer ezeket az utasításokat:

CLEARSCREEN, röviden CS;
CLEARTEXT;
TEXTSCREEN;
FULLSCREEN;
SPLITSCREEN.

Ha grafikus üzemmódban vagyunk, akkor a CLEARTEXT utasítás csak az osztott képernyő alsó négy szövegsorát törli. A Logo rendszer egyidejűleg és külön tárolja a képernyő szöveges és grafikus részének tartalmát. Bármikor hozzáférhetünk a két tartalom bármelyikéhez, ha beírjuk a TEXTSCREEN utasítást vagy pedig egy grafikai utasítást (tehát vagy a FULLSCREEN-t vagy a SPLITSCREEN-t). Látunk majd példát arra, hogy miképpen lehet gyorsan váltani a szöveges és grafikus üzemmód között.

Eddig a képernyőt szöveges üzemmódban úgy használtuk, hogy egyik sort a másik után írtuk be. A képernyő sorai ezután olykor felfelé kifutottak (scrolling). Egyes feladatok esetében kívánatos, hogy űrlapszerű, rögzített képernyőtartalom (képernyőmaszk) legyen látható. A maszk meghatározott helyein feltűnik vagy letörlődik a szöveg. Ez csak akkor lehetséges, ha vannak olyan utasításaink, amelyek a kurzort a kívánt képernyőpozícióra helyezik. Szöveges üzemmódban a kurzort a SETCURSOR utasítással lehet a képernyőn elhelyezni. A SETCURSOR bemenete egy lista, amelynek első eleme az oszlopot (0...39), második eleme pedig a sort (0...23) adja meg. A CURSOR funkcióval már a 2.8. szakaszban megismerkedtünk. A CURSOR a kurzor pillanatnyi helyzetét adja meg és így tabulálási funkciókban használható.

A következőkben bemutatunk néhány építőelemet, amelyeket a későbbi programokban eredményesen alkalmazhatunk.

Az ODA funkció a képernyő előre megadott helyére nyomtat. Eközben a TYPE CHAR 7 utasítás hatására rövid csöngetést kapunk.

```
?
TO "ODA
TO ODA :SZOVEG :HOVA
SETCURSOR :HOVA
TYPE CHAR 7
TYPE :SZOVEG
END
```

A TOROL funkció törli valamely sor megadott részét.

A TOROL első bemenete a kurzor helyzete, majd a törlőjel (szóköz vagy más jel) következik, a végére pedig a törlendő sorrész kerül. A TOROL használja a VONAL funkciót is, amely a kurzor helyzetének beállítása után megfelelő számú jelet nyomtat ki.

```
?
?
?
?TO TOROL :HOL :JEL :L
>SETCURSOR :HOL
>VONAL :L :JEL
>END
TOROL DEFINED
?

?
?TO VONAL :L :JEL
>REPEAT :L [TYPE :JEL]
>END
VONAL DEFINED
?
?
```

A Logo rendszerben megoldható, hogy egyes jelek villogjanak vagy invertálva (fehérre feketével) jelenjenek meg. Az adott jelet az ASCII kódtáblázat alapján lehet átszámítani.

INVERZ a beírt szót invertált alakban nyomtatja ki. A program bekérése előtt a kurzor helyét egy SETCURSOR-ral meg kell adnunk.

```
?
?TO SZOVEG.INVERZ :SZOV
>LABEL "START
>IF EMPTYP :SZOV [STOP]
>TYPE INVERZ FIRST :SZOV
>MAKE "SZOV BF :SZOV
>GO "START
>END
SZOVEG.INVERZ DEFINED
?
TO INVERZ :JEL
TEST ASCII :SZOV > 64
IFT [UP CHAR (128 + ASCII :JEL)]
IFF [UP CHAR (64 + ASCII :JEL)]
END
```

VILLOG felépítésében teljesen hasonlít az INVERZ-hez, eltérés csak az átszámításban található.

```
TO SZOV.VILLOG :SZOV
LABEL "START
IF EMPTYP :SZOV [STOP]
TYPE VILLOG FIRST :SZOV
MAKE "SZOV BF :SZOV
GO "START
END

TO VILLOG :JEL
TEST ASCII :JEL < 64
IFT [OP CHAR (192 + ASCII :JEL)]
IFF [OP CHAR (128 + ASCII :JEL)]
END
```

Később egy olyan programot mutatunk be, amely a MASZK építőelem segítségével tetszőleges képernyőmaszkokat készít, és az itt bemutatott segédprogramokat azonos vagy kissé módosított alakban használja.

3.4. Az adatbevitel módjai

Az előző fejezetekben az adatokat funkcióbemenetekkel vagy a MAKE utasítással definiáltunk. A READCHAR és a READLIST funkció további lehetőséget nyújt. Ezek segítségével a felhasználó a program futása során viheti be az adatokat. A READLIST rövidítése RL, a READCHAR-é RC.

A READLIST egy olyan művelet, amely a billentyűzetről bevitt adatokat listaként adja tovább. A READLIST használatakor a kurzor továbbra is villog, és így jelzi a képernyőnek azt a pontját, ahol a bemeneti adatok meg fognak jelenni.

Ha csak egy szót akarunk bevinni, akkor FIRST READLIST-et kell használni.

A READCHAR művelet egyetlen betűt ad, amely az éppen megnyomott billentyűnek felel meg. A READLIST-tel ellentétben itt nem kell használni a RETURN gombot.

A következő példában egy adott nyomógombot kell kitalálni. Csak a gomb jelent helyes megoldást. Minden egyéb esetben a betűt "HIBAS" nyomógomb útmutatással kinyomtatjuk. A WAIT várakozó utasítás hatására a Logo egy másodpercig vár, mielőtt a következő Logo utasítást végrehajtaná, így időt hagy az olvasásra.

```
TO KEZDES
CLEARTEXT
PR [MELYIK A START NYOMOGOMB?]
TYPE [NYOMJ MEG EGY GOMBOT...:]
IF JOGOMB RC [STOP]
KEZDES
END
```

```
?
?
TO JOGOMB :GOMB
IF :GOMB = "# [PR [HELYES!] OP "TRUE]
(PR :GOMB [HIBAS!]) WAIT 60
OP "FALSE
END

?
?
?KEZDES
MELYIK A START NYOMOGOMB?
NYOMJ MEG EGY GOMBOT...:F HIBAS!
MELYIK A START NYOMOGOMB?
NYOMJ MEG EGY GOMBOT...:3 HIBAS!
MELYIK A START NYOMOGOMB?
NYOMJ MEG EGY GOMBOT...:# HIBAS!
MELYIK A START NYOMOGOMB?
NYOMJ MEG EGY GOMBOT...:HELYES!
?
```

A READCHAR művelettel a legkülönbözőbb programok valósíthatók meg. Adatrögzítésnél pl. ellenőrizni lehet a billentyűzetet, hogy nem történt-e hibás kezelés. A MASZK nevű programban ezt később látni fogjuk. A következő szakaszban pedig arra mutatunk példát, hogy mindegyik billentyű más-más funkciót mozgósít.

Fontos vizsgáló szó még a KEYP, amely azt vizsgálja, hogy egy adott nyomógombot megnyomtak-e. A kezelő számára megszabhatunk pl. egy időtartamot, amelyben a bemeneti műveleteket végezheti. Ha sor került rá, akkor továbblép, ellenkező esetben csönget. Ezzel vizsgálhatjuk meg azt is, hogy három másodpercen belül ki tud több billentyűt leütni. A billentyűzet REPEAT nyomógombját azonban nem szabad használni! Lássuk ezt a kis programot:

```
?
?TO TESZT
>PR [ADATBEVITEL]
>PR [A SZAMITOGEP 3 MASODPERCET VAR]
>WAIT 180
>TEST KEYP
>IFT [PR PUFFERTARTALOM]
>IFF [EBRESZTES TESZT]
>END
TESZT DEFINED
?
?
?TO PUFFERTARTALOM
>TEST KEYP
>IFF [OP "6 ]
>IFT [OP WORD READCHAR PUFFERTARTALOM]
>END
PUFFERTARTALOM DEFINED
?
```

```
?
?TO EBRESZTES
>REPEAT 10 [TYPE CHAR 7 PR [NE ALUDJ!]]
>END
EBRESZTES DEFINED
?

?TESZT
ADATBEVITEL
A SZAMITOGEP 3 MASODPERCET VAR
NE ALUDJ!
NE ALUDJ!
NE ALUDJ!
NE ALUDJ!
NE ALUDJ!
NE ALUDJ!
NE ALUDJ!
NE ALUDJ!
NE ALUDJ!
NE ALUDJ!
ADATBEVITEL
A SZAMITOGEP 3 MASODPERCET VAR
NE ALUDJ!
NE ALUDJ!
NE ALUDJ!
STOPPED!:
 JUST BEFORE LEAVING EBRESZTES
?
?
```

A PUFFERTARTALOM funkció meghatároz minden olyan betűt, amelyet begépeltünk.

Befejezésül ismerkedjünk meg a NENYULJHOZZAM nevű programmal! A program egy végtelen ciklus, amely azt vizsgálja, hogy hozzányúlt-e valaki a billentyűzethez. Ha igen, akkor a NENYULJHOZZAM káromkodást ír ki. Ha nem, akkor a NENYULJHOZZAM a HORKOLNI funkció szerint működik.

```
?
?
?
?TO NENYULJHOZZAM
>TEST KEYP
>IFT [KAROMKODNI TOROLNI]
>IFF [HORKOLNI]
>NENYULJHOZZAM
>END
NENYULJHOZZAM DEFINED
?
?
?TO KAROMKODNI
>CLEARTEXT
>SETCURSOR [0 0]
```

```
>LZ 7 PR "A FENE EGYE MEG!
>LZ 3 PR "KI PISZKAL ENGEM???
>LZ 3 PR "HAGYD ABBA!!! EL A KEZEKKEL!!
>VILLOGNI
>END
KAROMKODNI DEFINED
?

?
?TO TOROLNI
>TEST KEYP
>IFT "ELFELEJT RC TOROLNI
>END
TOROLNI DEFINED
?
?
?TO HORKOLNI
>PR "HORKOLOK
>WAIT 60
>END

?
?TO ELFELEJT :S
>END
ELFELEJT DEFINED
?
?
?TO LZ :N
>REPEAT :N "PR "6  
>END
LZ DEFINED
?
?
?
?PO "VILLOGNI
TO VILLOGNI
HIDETURTLE
REPEAT 3 "FULLSCREEN WAIT 20 TEXTSCREEN WAIT 30
TEXTSCREEN
CLEARTEXT
END

?
?
?NENYULJHOZZAM
HORKOLOK
HORKOLOK
HORKOLOK
HORKOLOK



A FENE EGYE MEG!


KI PISZKAL ENGEM???
```

```
HAGYD ABBA!!! EL A KEZEKKEL!!
HORKOLOK
HORKOLOK
HORKOLOK
STOPPED!:
 JUST BEFORE LEAVING HORKOLNI
?
?
```

A VILLOGNI hatására a képernyőn levő szövegek azért villognak, mert felváltva aktivizálódik a grafikus és a szöveges képernyő. A bemeneti puffert TOROLNI-vel üríthetjük ki, amely mindaddig önmagát hívja, amíg a puffer ki nem ürül. Az ELFELEJT egy „nullfunkció", semmit sem tesz. Csupán azért szükséges, mert RC-vel mindig felveszünk egy újabb jelet a bemeneti pufferből. Bár ezt úgy is megoldhatjuk, hogy a felvett betűt mindig ugyanahhoz a lokális változóhoz rendeljük.

3.5. Vezérlő szerkezetek

3.5.1. A Logoban alkalmazható vezérlő szerkezetek

A könyv első részében a Logo szinte valamennyi vezérlő szerkezetével megismerkedtünk. Egy- és kétágú programelágazásokat írtunk, amelyeket az IF, TEST, az IFT és/vagy IFF szavakkal definiáltunk.

Az ismétléseket különbözőképpen fogalmaztuk meg. Vagy a REPEAT utasítást használtuk, vagy rekurzív funkciókat írtunk fel. A GO utasítás, a belépési címkével és a leállási feltétellel a programciklus összeállítására egy újabb típus volt.

A rekurzív funkciókat vagy valamilyen nyomtatóutasítás, vagy a STOP állította le. A rekurzív műveletek írásakor a kötelező OUTPUT-ot alkalmaztuk, mivel az előző hívófüggvénynek egy értéket tovább kellett adni. A rekurzív utasításokban a STOP utasítás végrehajtására akkor került sor, ha a leállási feltétel teljesült.

Ismerjük már a RUN utasítást is. A RUN a következő fejezetben lesz fontos, ahol saját vezérlő szerkezeteket állítunk elő. Tekintsük át még egyszer az ismert szerkezeteket és a hozzájuk tartozó utasításokat.

Egy- és kétágú elágazások:	IF [igaz-ág]
	IF [igaz-ág] [hamis-ág]
	TEST
	IFTRUE [igaz-ág]
	IFFALSE [hamis-ág]
Leállítás:	OUTPUT
	STOP

Többször ismétlés:	REPEAT <szám> <utasítás>.
Ugrási utasítás:	GO <belépési címke> LABEL <belépési címke>
Egy utasítássor kiértékelése és végrehajtása:	RUN.

A funkciók leállításának további lehetősége a THROW "TOPLEVEL alkalmazása. A "TOPLEVEL a közvetlen bemeneti üzemmódot jelenti. A következő példákkal megvilágítjuk a TOPLEVEL és a STOP különbségét.

```
?
?TO B
>C
>PR [B PROGRAM VEGE]
>END
B DEFINED
?
?
?TO C
>PR [ITT A C PROGRAM]
>THROW "TOPLEVEL
>END
C DEFINED
?
?
?
?B
ITT A C PROGRAM
?
```

A B program a C programot hívja. Az utóbbiban a második utasítássor kikényszeríti a TOPLEVEL állapotot. A hatása hasonló a CTRL-G-hez, ami a feltétlen programmegszakítást jelenti. Ha a program második sorában STOP utasítás állna, akkor csak a C program futása szakadna meg, a B program tovább futna.

```
TO C
PR [ITT A C PROGRAM]
STOP
END

?
?B
ITT A C PROGRAM
B PROGRAM VEGE
?
```

Gyakran szükséges a tesztelés során, hogy a programok futását rövid időre megállítsuk. Ezt kétféle funkció segítségével oldhatjuk meg: vagy a WAIT-tel, vagy PAUSE-zal.

A WAIT bemenete egy szám, amely megadja, hogy az egyhatvanad másodperc hányszorosa legyen a szünet. Szüneteket azért iktatunk be, hogy a felhasználónak legyen ideje a képernyőn levő szöveget elolvasni, mielőtt további adatok jelennének meg. A WAIT segítségével a képernyő teljes tartalma villogtatható. Ehhez csak a szöveges üzemmód és a grafikus üzemmód között kell ide-oda kapcsolgatnunk, miközben egy kis szünetet hagyunk. Ha a grafikát akarjuk villogtatni, akkor előzetesen a szövegképernyőt kell törölni. Ha a szöveget, akkor fordítva. A következő példa ezt szövegesen szemlélteti.

```
?
?TO VILLOG
>URES.ERNYO
>MONDAT
>NYOLCAS 60
>REPEAT 20 [VARJ]
>END
VILLOG DEFINED
?

?TO URES.ERNYO
>CLEARSCREEN
>CLEARTEXT
>TEXTSCREEN
>SETCURSOR [0 0]
>END
URES.ERNYO DEFINED
?
?
?TO MONDAT
>REPEAT 12 [PR "ö ]
>PR [ö ö ö "EZ ITT EGY FEKVO NYOLCAS]
>END
MONDAT DEFINED
?
?
TO NYOLCAS :SUGAR
CIRCLER :SUGAR
CIRCLEL :SUGAR
END

?
?TO VARJ
>TEXTSCREEN
>WAIT 10
>FULLSCREEN
>WAIT 10
>END
VARJ DEFINED
?
?
```

Ha a WAIT értéke háromnál kisebb, akkor az emberi szem a váltást már nem tudja követni, ilyenkor csak „egy" képet látunk.

A PAUSE hatására a program szintén megáll. A PAUSE után azonban a felhasználónak kell a további feldolgozást megindítani, mégpedig a CO (CONTINUE) utasítással. CTRL-Z-vel a billentyűzetről bármikor megszakítható a futás, így azonban nehéz egy meghatározott helyen megállni. Erre csak a PAUSE képes. A PAUSE igen alkalmas arra, hogy nagyobb programok „belövési" időszakában ellenőrző pontokat építsünk be. Ezzel a funkcióparaméterek és más, lokális értékek megvizsgálhatók, vagy egyes értékek tesztelési célokból módosíthatók. A PAUSE után megváltozik a válaszjel. Válaszjelként a programnév és egy hozzá kapcsolódó kérdőjel jelenik meg mindaddig, amíg CO-val nem folytatjuk a futtatást. Ilyenkor kinyomtathatjuk a változók tartalmát, megvizsgálhatjuk, hogy a munkatárban milyen funkciók vannak, sőt más programokat is hívhatunk. A következőkben három példát mutatunk, amelyek mindegyikében szerepel a PAUSE és hatása.

```
?TEXTSCREEN
?
?
?
?TO X
>PR [ITT VAN X]
>PAUSE
>PR [X A PAUSE UTAN]
>END
X DEFINED


?
?
?TO Y :MONDAT
>PR [SZIA ITT VAN Y]
>PAUSE
>PR [Y A PAUSE UTAN]
>END
Y DEFINED
?
?
?TO Z
>PR [PROG Z FUNKCIONEVEKET IR KI]
>POTS
>END
Z DEFINED
?
?
?X
ITT VAN X
PAUSING... IN X:
PAUSE
X ?CO
X A PAUSE UTAN
?
```

```
?X
ITT VAN X
PAUSING... IN X:
PAUSE
X ?PO "X
TO X
PR [ITT VAN X]
PAUSE
PR [X A PAUSE UTAN]
END

?Y [A B C]
SZIA ITT VAN Y
PAUSING... IN Y:
PAUSE
Y ?PR :MONDAT
A B C
Y ?PR :KAPU
KAPU HAS NO VALUE
Y ?X
ITT VAN X
PAUSING... IN X:
PAUSE
X Y ?CO
X A PAUSE UTAN
Y ?Z
PROG Z FUNKCIONEVEKET IR KI
TO Z
TO Y :MONDAT
TO X
TO VARJ
TO NYOLCAS :SUGAR
TO MONDAT
TO URES.ERNYO
TO VILLOG
TO C
TO B
Y ?CO
Y A PAUSE UTAN
?
```

3.5.2. Más programnyelvek vezérlő szerkezetei a Logoban

Aki foglalkozott már más programnyelvekkel, az alighanem megszeretett néhány vezérlő szerkezetet. Például a FOR-NEXT ciklus nélkül a BASIC-et el sem tudjuk képzelni. Az esetkiválasztás (CASE... OF...) és a WHILE vagy REPEAT...UNTIL adta ismétlési lehetőségek kényelmesek, és nem szívesen mond le az ember az ismert munkamódszerekről. A következő pontokban saját vezérlő szerkezetet definiálunk, amelyek felhasználásával a programozási munka gyorsítható és ésszerűvé tehető.

A WHILE–DO utasítás

A WHILE–DO utasításnak két bemenete van. Az első az ismétlési utasítás leállási feltétele, a második a programnév vagy a kívánt utasítássorok.

```
?
?TO WHILE :FELTETEL :TEDD
>LABEL "START
>IF RUN :FELTETEL [RUN :TEDD] [STOP]
>GO "START
>END
WHILE DEFINED
?
?
?MAKE "LISTA [A B C]
?
?
?TO NYOMTASS
>PR FIRST :LISTA
>MAKE "LISTA BF :LISTA
>END
NYOMTASS DEFINED
?
?
?WHILE [NOT EMPTYP :LISTA] [NYOMTASS]
A
B
C
?
```

Magyarul ez a ciklusutasítás a közvetkezőképpen hangzik: MIG (a definiált feltétel teljesül) TEDD (... ezt és ezt).

Aki a Logo programokban látni szeretné a DO szócskát vagy a magyar nyelvű változatban a "TEDD" szócskát, annak csak meg kell írnia a DO vagy "TEDD" nullműveleteket, és máris teljesült a kívánsága.

```
TO MIG :FELTETEL :TEDD
LABEL "START
IF RUN :FELTETEL [RUN :TEDD] [STOP]
GO "START
END

?
TO TEDD :§
OP :§
END

?
?MAKE "LISTA [A B C]
?MIG [NOT EMPTYP :LISTA] TEDD [NYOMTASS]
A
B
C
?
?
```

A REPEAT–UNTIL utasítás

Ezt a ciklusutasítást nem tekinthetjük egyszerűen a REPEAT Logo utasítás megfelelőjének. A REPEAT bemenete mindig egy szám, amelyet számláló ciklusnak is neveznek. A REPEAT–UNTIL ciklus általánosabb: a feltétele egy tetszőleges vizsgáló szó, és a számláló ciklusokhoz hasonlóan az alsó és felső értékhatárt vizsgálja.

A WHILE–DO utasítással ellentétben ezt a ciklustípust a gép egyszer mindenképpen végrehajtja, hiszen csak az utasítás végrehajtása után van a feltételvizsgálat. Az ismétlődés itt is addig tart, amíg a beírt feltétel teljesül.

```
?
?
?TO ISMETEL :TEDD :EDDIG
>LABEL "START
>RUN :TEDD
>IF NOT RUN :EDDIG [GO "START]
>END
ISMETEL DEFINED
?

?TO EDDIG :§
>OP :§
>END
EDDIG DEFINED
?
```

A FOR ciklus

A FOR ciklus egy számláló ciklus. A változó értékét rögzített lépéssel a végértékig változtatja. A változó minden egyes lépése után a ciklusmag utasításait vagy programjait a cilusváltozó pillanatnyi értékével hajtja végre a gép.

A FOR és a FOR1 programváltozat közül az utóbbi áttekinthetőbb. A REPEAT utasításos megoldás némileg gyorsítja is a végrehajtást.

```
?
?TO FOR :VAR§ :A§ :E§ :S§ :UTASITAS
>LOCAL :VAR§
>MAKE :VAR§ :A§
>REPEAT SUM 1 (:E§- :A§) / :S§ [RUN
>[:UTASITAS MAKE :VAR§ :S§ + THING :VAR§]
>END
FOR DEFINED
?
?
?TO FOR1 :VAR§ :A§ :E§ :S§ :UTASITAS
>LOCAL :VAR§
>MAKE :VAR§ :A§
>LABEL "START
>RUN :UTASITAS
```

```
 MAKE :VAR$ :S$ + THING :VAR$
>IF THING :VAR$  :E$ [STOP]
>GO "START
>END
FOR1 DEFINED
?
```

A példából kitűnik, hogy visszafelé is lépkedhetünk a számláló ciklusban, és kisebb értékváltoztatás is megengedett, vagyis a lépésköz 1-nél kisebb is lehet.

```
?
?FOR "I 60 65 1 [PR WORD CHAR :I CHAR :I]
<<
==
>>
??
$$
AA
?
?
?FOR "I 5 1 -1 [PR SQRT :I]
2.23607
2.
1.73205
1.41421
1.
?
?
?
?FOR "I 1.01 1.015 .001 [PR :I * 1000]
1010.
1011.
1012.
1013.
1014.
?
```

Utolsó példánkban a gép már nem hajtotta végre az utolsó utasítást. A REPEAT utasítás ui. egész számokkal dolgozik, tehát adott szám egész részét veszi, például az 5,99988-ból csak az 5-öt. Ha az osztás eredményét kerekítettük volna, akkor minden rendben lenne. A FOR utasítást tehát ki kell egészíteni a ROUND művelettel.

```
?
?PR (1.015-1.01) / .001
4.99988
?
?PR ROUND (1.015-1.01) / .001
5
?
```

Többszörös elágazás CASE-OF-fal

Eddig az egy- és kétágú programelágazást ismertük. Most bemutatunk egy olyan vezérlő szerkezetet, amely lehetővé teszi a többszörös elágazást. Ebben a Logo változatban nem szerepel az igen hasznos MEMBER alapfunkció, amely a CASE–OF vezérlő szerkezet fontos alkotóeleme. A MEMBER hasonlóan működik, mint a már bemutatott ITEM funkció, amely a lista egy számmal megadott elemét választja ki. A MEMBER a megadott elemtől kezdve sorolja fel a lista elemeit.

```
?
?PR MEMBER "C [A B C D E]
C D E
?
?PR MEMBER "3 "SZO5673A
3A
?
?PR FIRST BF MEMBER "B [A EN B TE C D]
TE
```

Lássuk tehát a CASE–OF vezérlő szerkezetet, amelyben a MEMBER-t rekurzívan hívjuk meg.

```
?
?
?TO MEMBER :S :SOBJ
>IF EMPTYP :SOBJ [OP :SOBJ]
>IF :S=FIRST :SOBJ [OP :SOBJ]
>OP MEMBER :S BF :SOBJ
>END
MEMBER DEFINED
?
?
?TO CASE :ESET :ESETEK
>OP FIRST BF MEMBER :ESET :ESETEK
>END
CASE DEFINED
?
?
```

A CASE–OF szerkezettel tetszőleges esetkombinációkat hozhatunk létre és a beadott egyedi esettől függően bármerre elágaztathatunk. A listában szereplő esetek szavak vagy listák lehetnek. Itt különösen célszerű listákat használni, mivel ezek kinyomtathatók, vagy a RUN utasítás bemenetei lehetnek. Így tehát a CASE–OF szerkezettel különböző adatokra ágaztathatunk el, amelyek vagy programadatként vagy programként értelmezhetők.

A következő példák némileg megvilágítják ezt.

```
?
?TO IGOMB
>PR [IGOMB PROGRAM FUT]
>END
IGOMB DEFINED
?
?
?TO KGOMB
>PR [KGOMB  PROGRAM FUT]
>END
KGOMB DEFINED
?
?
?TO JGOMB
>PR [JGOMB PROGRAM FUT]
>END
JGOMB DEFINED
?
?
?PR CASE "J [I [IGOMB] J [JGOMB] K [KGOMB]]
JGOMB
?
?
?
?RUN CASE "J [I [IGOMB] J [JGOMB] K [KGOMB]]
JGOMB PROGRAM FUT
```

A CASE–OF szerkezetnek rendszerint hibakimenete is van, amelynek segítségével hibás esetben a program összeomlása elkerülhető. Az előre nem látott vagy nem végiggondolt eseteket így lehet kezelni. A következő, módosított program ezt a járulékos lekérdezést tartalmazza. Itt a HIBA programként értelmezhető, és a mi feladatunk, hogy pontosabban definiáljuk.

```
TO CASE :ESET :ESETEK
IF NOT MEMBERP :ESET :ESETEK [OP [HIBA]]
OP FIRST BF MEMBER :ESET :ESETEK
END

?
?TO HIBA
>OP [GO "CIKLUS]
>END
HIBA DEFINED
?
```

Teknőc-ábra, nyomógombokkal

Az elágaztatásra egy valósabb programot mutatunk. A billentyűzet egyes gombjaitól függően ágazunk el más és más programokba. Készítsünk egy programot, ami a billentyűkről vezérli a teknőcöt! Az egyes nyomógombok

egy bizonyos grafikai utasítást aktivizáljanak. Így hat elemi utasítással mint egy fényceruzával lehet geometriai ábrákat rajzolni. A következő utasításokat használjuk:

- a képrenyő törlése és a teknőc kiindulási helye;
- fordulás jobbra (10-10°-kal);
- fordulás balra (10-10°-kal);
- előre (10-10 lépéssel);
- hátra (10-10 lépéssel);
- a hibás mozgások kitörlése.

Ezekkel a grafikai utasításokkal már sokféle ábra rajzolható.* A következő billentyűket használjuk fel:

- B Balra forgás;
- J Jobbra forgás;
- E Előre;
- H Hátra;
- T képernyőTörlés;
- R Radírozás.

A hibás gombok elleni védelmet a CASE–OF szerkezet hibakimenete adja. Amennyiben a begépelt betű nem szerepelt az esetek listájában, kimenetként a GO „ciklus" utasítást kapjuk. A hibás bemenetet ily módon a gép figyelmen kívül hagyja anélkül, hogy a felhasználó ezt észrevenné.

```
?
?TO GRAFIKA
>LABEL "CIKLUS
>RUN CASE RC UTASITASOK
>GO "CIKLUS
>END
GRAFIKA DEFINED
?
?
?TO UTASITASOK
>OP [B [BALRA] J [JOBBRA] E [ELORE] H [HATRA] T
>[TORLES] R [RADIROZAS]]
>END
UTASITASOK DEFINED
?
?
?TO RADIROZAS
>PENERASE
>BK 10
>PENDOWN
>END
RADIROZAS DEFINED
?
```

* Még változatosabbak lennének az ábrák, ha változóként adnák meg a lépéshosszt, ill. a szögelfordulást (Lektor).

```
)
?TO HATRA
>BK 10
>END
HATRA DEFINED
?
?
?TO TORLES
>CS
>END
TORLES DEFINED
?
?
?TO ELORE
>FD 10
>END
ELORE DEFINED
?
?
?TO BALRA
>LT 10
>END
BALRA DEFINED
?
?

?TO JOBBRA
>RT 10
>END
JOBBRA DEFINED
?
?
?
TO CASE :ESET :ESETEK
IF NOT MEMBERP :ESET :ESETEK [OP HIBA]
OP FIRST BF MEMBER :ESET :ESETEK
END

?
```

A menümódszer

Ismerkedjünk meg az útbaigazító menütechnikával. Ebben csak az adott lehetőségek közül lehet választani, a többi nyomógomb védve van a gépelési hibák ellen. Egy menü pl. a következő:

```
MELYIK LECKET KERI:

                TEKNOSBEKA GR.....A

                CSILLAGKOD........B

                REPEAT - UTASITAS.C

                BEFEJEZES.........D
```

A megfelelő betű vagy szám begépelése után a program elágazik. A menütechnika sokféle helyen használható, játékok kiválasztására, bonyolult adatrögzítési programokban segédrutinok hívására stb. Az alapelv minden esetben azonos. A következő program szerkezetét mutatjuk először be:

```
?
TO LOGOTANFOLYAM
LABEL "CIKLUS
MENUNYOMTATAS
VALASZTAS
GO "CIKLUS
END

TO MENUNYOMTATAS
CLEARTEXT
SETCURSOR [0 0]
REPEAT 4 [PR " ]
PR [MELYIK LECKÉT KERI:]
PR [] TAB 15 PR [TEKNOSBEKA GR.....A]
PR [] TAB 15 PR [CSILLAGKOD.....B]
PR [] TAB 15 PR [REPEAT - UTASITAS.C]
PR [] TAB 15 PR [BEFEJEZES.....D]
END

TO TAB :HELY
REPEAT (:HELY - FIRST CURSOR) [TYPE " ]
END

TO VALASZTAS
LABEL "MELLENYULT
RUN ( SE CASE RC HOZZA MENU )
END

?
TO CASE :ESET :ESETEK
IF NOT MEMBERP :ESET :ESETEK [OP HIBA]
OP FIRST BF MEMBER :ESET :ESETEK
END

?
TO HIBA
PR CHAR 7 PR CHAR 7
OP [GO "MELLENYULT]
END

?
TO MEMBER :$ :$OBJ
IF EMPTYP :$OBJ [OP :$OBJ]
IF :$ = FIRST :$OBJ [OP :$OBJ]
OP MEMBER :$ BF :$OBJ
END

?
TO HOZZA :$
OP :$
END
```

```
?
TO MENU
OP (A ALECKE B BLECKE C CLECKE V VEGE)
END.

?
TO VEGE
THROW "TOPLEVEL
END
```

A program szerkezete:

LOGOTANFOLYAM
- MENUNYOMTATAS
  - TAB
- VALASZTAS
  - CASE
    - HIBA
    - MEMBER
  - HOZZA
  - MENU
    - VEGE

Megjegyezzük, hogy a választási lista kialakítása most eltér az előbbi pontétól. Itt a lista elemei ui. szavak, nem pedig listák. Mivel azonban a RUN bemenete kötelezően lista, ezért az esetkiválasztás eredményét még listává kellett alakítanunk a SENTENCE utasítással. Gömbölyű zárójelek azért szükségesek, mert a SENTENCE-nek csak egy bemenete van. Láthatjuk, hogy a CASE–OF szerkezettel kombinált RUN utasítássorok többféleképpen oldhatók meg.

Az előző vezérlő program maga is nagyobb programcsomag része lehet. Ha a mágneslemezről beolvasunk egy másik leckét, akkor az előző programok törlődnek. Így elérhető, hogy a munkatár sose legyen túl kicsi. Ehhez a LOGOTANFOLYAM funkcióit a következőképpen kell módosítani:

```
TO LOGOTANFOLYAM
LABEL "CIKLUS
MENUNYOMTATAS
VALASZTAS
ERPS
GO "CIKLUS
END

?
?PO "VALASZTAS
TO VALASZTAS
LABEL "MELLENYULT
RUN DISZKROL.BEHOZNI CASE RC HOZZA MENU
START
END
```

```
TO DISZKROL.BEHOZNI :ADATI
IF :ADATI = "VEGE [VEGE]
IF :ADATI = [HIBA] [OP HIBA]
OP SE [LOAD] WORD "  " :ADATI
END

?PO "CASE
TO CASE :ESET :ESETEK
IF NOT MEMBERP :ESET :ESETEK [OP HIBA]
OP FIRST BF MEMBER :ESET :ESETEK
END

?
```

Így a LOGOTANFOLYAM valamennyi funkciója a tárban láthatatlanná van téve, az ERPS törlő utasítás csak a nem kívánt leckét törli, helyet adva az új lecke számára. Az a kiválasztott eset aDISZKROL.BEHOZNI függvény bemenete. Az utolsó sor egy listát állít elő, ami a kívánt LOAD utasítást tartalmazza. Ezt később a RUN végrehajtja.

A VALASZTAS utolsó utasítássora a START. Ez a lemezről betöltött, minden egyes programcsomag irányító programja.

3.5.3. Hibakimenetek definiálása

A Logo vezérlő szerkezetei közé tartozik még a CATCH és THROW utasítás. A CATCH és a THROW egy utasításpárt alkot, hasonlóan a GO és LABEL utasításpárhoz. A THROW átadja (átdobja) a programvezérlést a hozzá tartozó CATCH-nek. Az összetartozást a közös név adja, ez a CATCH és a THROW bemenete. Amennyiben egy felhasználói funkcióban előfordul a THROW "HIBA utasítás, akkor a gép megkeresi a hozzá tartozó CATCH "HIBA jelet, amelyet valamilyen korábbi részprogramban kell definiálni. CATCH-csel és THROW-val tehát saját programkimeneteket definiálhatunk. Így tehát kiugorhatunk a programokból és a teljes program korábbi részébe visszaléphetünk.

Lássuk ezt egy példa alapján! Az A...E programokban nyomtatási utasításokat adunk, amelyek lehetővé teszik a futás nyomon követését. A D programban közvetlen visszaugrás van az A programba, E a program vége.

```
?
?
?TO A
>CATCH "B [B]
>PR [AZ A PROG UTOLSO SORA]
>END
A DEFINED
?
?
TO B
PR [B HIVJA C - T]
C
END
```

```
?
?
?TO C
>PR [MOST HIVOM D-T]
>D
>END
C DEFINED
?
?
?TO D
>PR [ITT A D]
>PR [VISSZA AZ A-BA VAGY AZ E-BE?]
>TEST FIRST RL = "A
>IFF [E]
>PR [VISSZA A-BA THROW "B-VEL]
>THROW "B
>END
D DEFINED
?
?
?TO E
>PR [PROGRAM VEGE]
>THROW "TOPLEVEL
>END
E DEFINED
?
?
```

Futtassuk le az A tesztprogramot, és lássuk, hogy mi lesz az eredmény a D bemenete függvényében:

```
?A
B HIVJA C - T
MOST HIVOM D - T
ITT A D
VISSZA AZ A - BA VAGY AZ E - BE?
A
VISSZA A - BA THROW "B - VEL
AZ A PROG UTOLSO SORA
?
?
?A
B HIVJA C - T
MOST HIVOM D - T
ITT A D
VISSZA AZ A - BA VAGY AZ E - BE?
E
PROGRAM VEGE
?
```

Fontos, hogy a CATCH második bemenete utasításlista legyen, amit a THROW-ról CATCH-ra ugorva a program végrehajt.

A THROW két speciális bemenete a "TOPLEVEL, és az "ERROR. THROW "TOPLEVEL hatására a program futása azonnal megszakad, tehát a hatása megegyezik egy azonnal begépelt CTRL-G-vel. A THROW "ERROR-t

a Logo rendszer automatikusan végrehajtja, ha valamilyen hiba bukkan fel. Ha nincs megfelelő CATCH "ERROR – amit a felhasználó definiált –, akkor a Logo rendszer a szokásos hibaüzenetet nyomtatja ki és vár. Vizsgáljuk meg részletesebben, hogy mi történik akkor, ha a felhasználói funkcióba beépítünk egy ilyen CATCH "ERROR-t?

A LEMEZRE feladata, hogy a munkatár tartalmát kimentse és az azonos nevű fájlokat előzetesen törölje. Emlékezzünk vissza arra, hogy már létező fájlok esetében az ERASEFILE az egyetlen lehetőség arra, hogy a módosított fájlt ugyanezen a néven ismét tároljuk.

```
?
?TO LEMEZRE :FILE
>ERASEFILE :FILE
>SAVE :FILE
>END
LEMEZRE DEFINED
```

Nem szűrtük még ki azonban azt az esetet, amikor nem létező fájlt akarunk kimenteni.

```
?
?LEMEZRE "DUDA
FILE NOT FOUND IN LEMEZRE:
ERASEFILE :FILE
?
```

Módosítsuk tehát a programunkat, építsük be a CATCH "ERROR-t: A THROW "ERROR-t a CATCH "ERROR elkapja, és a közvetlenül utána álló utasítássor végrehajtódik. Mivel most a SAVE utasítás következik, elképzelésünknek megfelelően a munkatár tartalma a lemezre kerül.

A hibakapcsolókat arra is használhatjuk, hogy letiltsuk az angol hibaüzeneteket és helyette sajátot írassunk ki.

```
?
?TO ERF :FILE
>CATCH "ERROR [ERASEFILE :FILE]
>IF EMPTYP ERROR [STOP]
>PRINT CHAR 7
>PR [MELYIK FILE-T TOROLJEM?]
>(PR [A FILE] :FILE [NEM LETEZIK!])
>PR [UJRA]
>END
ERF DEFINED
?
?
```

Az ERROR Logo funkció olyan művelet, amely a hiba fellépése után hibaleírást ad. Normális esetben, ha nem történt hiba, az ERROR művelet eredménye üres lista. Mivel hiba esetén ERROR a megfelelő hibaleírásokat adja, az ERF funkció nem fejeződik be, hanem figyelmeztet.

A SYNTAXCHECKER rekurzív, mindig a bemeneti sort hajtja végre. Szintaktikai hiba esetén csönget és közli a hibát.

```
?
?TO SYNTAXCHECKER
>CATCH "ERROR [RUN RL]
>IF NOT EMPTYP ERROR [TYPE CHAR 7 PR "HIBA]
>SYNTAXCHECKER
>END
SYNTAXCHECKER DEFINED
?
?
```

A SYNTAXCHECKER-t természetesen ki lehet bővíteni, hogy a kezdők számára az ERROR elemzése alapján részletes magyar nyelvű útmutatást adjon. Egyébként ezt a programot CTRL-G-vel nem tudjuk megszakítani, erre csak a CTRL-Z és a THROW "TOPLEVEL utasítás alkalmas.

A CATCH "ERROR-ral (ill. "HIBA-val) kapcsolatban még megjegyezzük, hogy az általunk használt Logo változatban a hozzá kapcsolt utasításlista csak akkor működik megbízhatóan, ha az utasítások egyszerű utasítások. Ha műveleteket adunk meg, akkor hiba léphet fel (l. a 3.6. fejezetben levő UTOD példát).

3.6. Strukturált listák

Ez a fejezet elsősorban azokat a lehetőségeket mutatja be, amelyek a strukturált listákban és a tulajdonságlistákban rejlenek. A harmadik fő részben szereplő programpéldák szinte kizárólag strukturált listákkal vagy olyan listákkal dolgoznak, amelyek összefüggenek egymással.

Eddig olyan listákkal foglalkoztunk, amelyeknek elemei szavak voltak. Az 1. fejezetben már utaltunk arra, hogy a lista eleme további lista is lehet. Lássuk a háromelemű lista egyszerű példáját:

[[PULI TACSKO] [SZIAMI PERZSA] [KACSA GALAMB]].

Minden egyes elem egy-egy kételemű listából áll, amelynek elemei szavak. Az ilyen listát másodrendű listának nevezzük. Amennyiben a listaelem szavak helyett listákat tartalmaz, akkor harmadrendű listával van dolgunk.

Számos jelenség jól írható le strukturált listák segítségével. Ennek során a listaelemek sorrendje és maguk az elemek is jelentősek.

Másodrendű lista a rendezett számpárok sorozata. Az egyes számpárok a függvénygörbe x és y koordinátái, ill. egy képernyő képpontjai lehetnek:

[[-10 12] [-5 7] [0 3] [5 7] [10 30]].

Egy további példa egy öt sorból és öt oszlopból álló táblázat, vagyis egy 5X5 elemből álló mező:

[[A 3 5 EN 7] [...] [...] [...] [...]].

A táblázat öt oszlopelemből álló első sora: A, 3, 5, EN, 7. A többi négy sort csak jelöltük.

Amennyiben a sorban levő mezőelemek szavak, akkor másodrendű listával van dolgunk. A sorelemek azonban ismét lehetnek listák. A táblázat minden egyes mezőeleme kételemű listaként pl. egy számpárt tartalmaz.

Harmadik példaként lássuk a következő listaelemet:

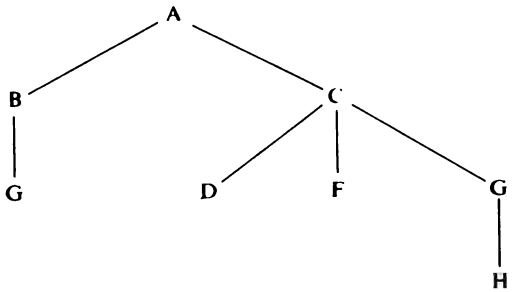
[[5 10] HELY VILLOGNI].

Ez a listaelem azt jelöli, hogy a 10. képernyősor az 5. pozíciótól kezdve villogva tartalmazza a "HELY nevű változó tartalmát.

Lássunk egy negyedik példát is:

[A [B C] B [G] C [D F G] D[] F[] G [H]].

Ez a lista egy gráfnak tekinthető, ami pl. egy felhasználói program hívási szerkezete lehet úgy, hogy az A nevű főprogram hívja a B és C felhasználói funkciókat stb.



Hasonlóan hierarchikus szerkezetű a cégek szervezési vázlata, a családfák, az osztályozások, a hálótervek. Ezek a listák bizonyos pontokon további mutatókat tartalmazhatnak, amelyek egy előző vagy egy későbbi változóra utalnak, és így adatláncokat írnak le. Ezek a mutatók más fájlok adatainak azonosító számai is lehetnek. Ily módon meglehetősen bonyolult adatszerkezetek is leírhatók. Ilyen adatok feldolgozására megfelelő felhasználói funkciókat kell írni.

Ismerkedjünk meg a tulajdonságlistákkal is! Ehhez az adattípushoz a Logo egy sereg utasítást kínál, ezért tehát könnyen kezelhető. A tulajdonságlisták fogalmának tisztázásához visszatérünk e fejezet első példájához, amelyben

a háziállatokat soroltuk fel. A háziállatlista minden egyes eleme elé egy leíró szót tettünk: [KUTYAK [PULI TACSKO] MACSKAK [SZIAMI PERZSA] MADARAK [KACSA GALAMB]]:

Másképp fogalmazva azt mondhatjuk, hogy a "HAZI ALLATOK tulajdonságlista a "KUTYAK "MACSKAK és "MADARAK tulajdonságokat tartalmazza, amikor is minden egyes tulajdonságnak két értéke van, vagyis két példát tartalmaz. Ismerkedjünk most meg a tulajdonságlistákat kezelő Logo funkciókkal!

A PLIST (Property List) művelet kiírja azt a tulajdonságlistát, amelynek beírtuk a nevét.*

```
?
?SHOW PLIST "HAZIALLATOK
[KUTYAK [TACSKO PULI] MACSKAK [SZIAMI PERZSA]
MADARAK [GALAMB KACSA]]
```

A GPROP (Get Property) művelet megadja egy tulajdonságlista adott tulajdonságát és a tartalmát. A bemenet a tulajdonságlista és a tulajdonság neve.

```
?
?SHOW GPROP "HAZIALLATOK "MACSKAK
[SZIAMI PERZSA]
?
?SHOW GPROP "HAZIALLATOK "EGEREK
[]
?
```

A PPROP (Put Property) művelet a tulajdonságlistába beilleszt egy tulajdonságot, a tartalmával együtt. A bemenet a tulajdonságlista neve, a tulajdonság megjelölése és a tulajdonság tartalma. Ha a tulajdonságlista ezt a tulajdonságot már tartalmazza, akkor az új tulajdonság a tartalmával együtt lesz a lista aktuális tulajdonsága. A régi tulajdonság ezzel törlődik, ill. felülíródik.

```
?
?PPROP "HAZIALLATOK "JATEKALLATOK [HORCSOG
TENGERIMALAC]
?
?SHOW PLIST "HAZIALLATOK
[JATEKALLATOK [HORCSOG TENGERIMALAC] KUTYAK
[TACSKO PULI]
MACSKAK [SZIAMI PERZSA] MADARAK [GALAMB KACSA]]
?
?PPROP "HAZIALLATOK "JATEKALLATOK [PAPAGAJ]
?
?SHOW PLIST "HAZIALLATOK
[JATEKALLATOK [PAPAGAJ] KUTYAK [TACSKO PULI] MACSKAK

[SZIAMI PERZSA] MADARAK [GALAMB KACSA]]
```

* Innentől kezdve a listák többnyire szögletes zárójelben vannak.

A REMPROP (Remove Property) utasítás törli a tulajdonságlistákban levő tulajdonságokat, azaz törlődik a tulajdonság neve és tartalma.

```
?
?
?
?REMPROP "HAZIALLATOK "JATEKALLATOK
?
?SHOW PLIST "HAZIALLATOK
[KUTYAK [TACSKO PULI] MACSKAK [SZIAMI PERZSA]
MADARAK [GALAMB KACSA]]
?
```

A PPS (Print Out Properties) utasítás az összes tulajdonságot kilistázza. Ez érvényes a munkatár kezelésével kapcsolatos csomagkijelölésekre is (l. a 3.2. szakaszt).

```
?
?
?PPS
HAZIALLATOK'S KUTYAK IS [TACSKO PULI]
HAZIALLATOK'S MACSKAK IS [SZIAMI PERZSA]
HAZIALLATOK'S MADARAK IS [GALAMB KACSA]
AIDS'S BURY IS TRUE
.SYSTEM'S BURY IS TRUE
?
?
?
```

Ezeknek a funkcióknak és különösen a GPROP és a PPROP műveleteknek az az előnye, hogy a kijelölt listaelemet közvetlenül hívhatjuk, anélkül, hogy ismernünk kellene az elem helyét a listában. A tulajdonságlisták a Logoban külön adattípust képeznek, ezek az értékadási utasításokkal nem kezelhetők. SAVE és LOAD segítségével természetesen a meglevő tulajdonságlisták átvihetők.

A következőkben kidolgozunk néhány alapfunkciót, amelyekkel a láncolt tulajdonságlisták könnyen feldolgozhatók. A láncolás úgy történik, hogy a lista utódját vagy elődjét feljegyezzük a tulajdonságlistába.

Bemutatunk négy láncolt tulajdonságlistát. A szemléletesség kedvéért egymás utáni generációk listaelemei, a tulajdonság pedig hogy elő-e (E) vagy utód (U).

```
?
?PPROP "UKAPA "U "DEDAPA
?PPROP "DEDAPA "U "NAGYAPA
?PPROP "NAGYAPA "U "APA
?PPROP "APA "E "NAGYAPA
?PPROP "NAGYAPA "E "DEDAPA
?PPROP "DEDAPA "E "UKAPA
?
```

```
?
?SHOW PLIST "APA SHOW PLIST "NAGYAPA SHOW PLIST
"DEDAPA SHOW PLIST "UKAPA
[E NAGYAPA]
[E DEDAPA U APA]
[E UKAPA U NAGYAPA]
[U DEDAPA]
?
?
?PPS
APA'S E IS NAGYAPA
NAGYAPA'S E IS DEDAPA
NAGYAPA'S U IS APA
DEDAPA'S E IS UKAPA
DEDAPA'S U IS NAGYAPA
UKAPA'S U IS DEDAPA
AIDS'S BURY IS TRUE
.SYSTEM'S BURY IS TRUE
?
?
```

Most pedig kidolgozunk egy olyan felhasználói műveletet, amely meghatározza egy tulajdonságlista utódját:

```
?
?
?PRO "UTOD
TO UTOD :TOL
CATCH "ERROR [MAKE "TOL GPROP :TOL "U OP :TOL] OP []
END

?
?PR UTOD "NAGYAPA
APA
?
?SHOW UTOD "APA
[]
?
?SHOW PLIST UTOD "UKAPA
[E UKAPA U NAGYAPA]
?
```

Ezzel az alapfunkcióval további két funkciót definiálhatunk, amelyek a lista utódlistáinak számát és a lista n-edik utódját adják meg.

```
?
?PO "UTOD.SZAM
TO UTOD.SZAM :TOL
LOCAL "N
MAKE "N 0
LABEL "START
MAKE "TOL UTOD :TOL
IF EMPTYP :TOL [OP :N]
MAKE "N :N + 1
GO "START
END
```

```
?
?TO NIK.UTOD :TOL :N
>CATCH "ERROR [REPEAT :N [MAKE "TOL UTOD :TOL]]
>IF EMPTYP ERROR [OP :TOL] [OP []]
>END
NIK.UTOD DEFINED
?
?
?
?
?PR UTOD.SZAM "APA
0
?
?PR UTOD.SZAM "UKAPA
3
?
?SHOW NIK.UTOD "UKAPA 2
NAGYAPA
?
?SHOW NIK.UTOD "NAGYAPA 2
[]
?
?SHOW PLIST NIK.UTOD "UKAPA 3
[E NAGYAPA]
?
```

Hasonlóképpen definiálhatók az olyan funkciók, amelyek a tulajdonságlista elődjét dolgozzák fel (e keveréknyelv fordítása pl.: APA elődje NAGYAPA):

```
?
?PO "ELOD
TO ELOD :TOL
CATCH "ERROR [MAKE "TOL GPROP :TOL "E OP :TOL] OP []
END

?
?PO "ELOD.SZAM
TO ELOD.SZAM :TOL
LOCAL "N
MAKE "N 0
LABEL "START
MAKE "TOL ELOD :TOL
IF EMPTYP :TOL [OP :N]
MAKE "N :N + 1
GO "START
END

?
?PO "NIK.ELOD
TO NIK.ELOD :TOL :N
CATCH "ERROR [REPEAT :N [MAKE "TOL ELOD :TOL]]
IF EMPTYP ERROR [OP :TOL] [OP []]
END
```

```
?
?PR ELOD.SZAM "APA
3
?PR ELOD.SZAM "UKAPA
0
?SHOW NIK.ELOD "APA 2
DEDAPA
?SHOW PLIST ELOD "NAGYAPA
[E UKAPA U NAGYAPAJ
?
?
```

A funkciók érdekesen bővíthetők, ha olyan kereső funkciót készítünk, ami egy olyan adott utódot határoz meg, amely rendelkezik a bemenetként megadott tulajdonságokkal.

```
?
?
?
?TO KERES.UTOD :TOL :ISMERV
>LABEL "START
>MAKE "TOL KOVETK :TOL
>IF EMPTYP :TOL [OP []]
>IF MEMBERP LAST :ISMERV GPROP :TOL FIRST :ISMERV
 [OP :TOL]
>GO "START
>END
KERES.UTOD DEFINED
?
```

Az :ISMERV funkcióparaméter olyan lista, amelynek első eleme a tulajdonság neve és ezt követi a tartalma.

Ez a kereső funkció az adatállományt csak egy ismérv szerint vizsgálja. Csekély módosítással olyan kereső funkciókat is definiálhatunk, amelyek több szempontot vizsgálnak.

Ennek alapján további tulajdonságlistákkal bővíthetjük a kereső funkciókat, amelyek a tárolón levő nagyobb adatállományokat vizsgálják. A listák eredményeit további ismérvekkel bővíthetjük. Például az autósokról összeállított adatállományba felvehetjük a tulajdonost, a vezetőt, a forgalmi enegedélyt, a gépkocsi típusát, az ellenőrzés időpontját és a név kiegészítéseként a személyi számot. Attól függően, hogy milyen adat alapján keresünk, más-más kereső funkció hozhatja fel az összetartozó adatokat.

3.7. A programok programokkal dolgoznak

Eddig a funkciókat programokká állítottuk össze, ezek a programok szavakat és listákat dolgoztak fel. Azonban a Logoban definiált funkciók maguk is adatoknak tekinthetők, amelyek a szokásos módon mint listák és szavak feldolgozhatók. Erre a célra a TEXT és DEFINE alapfunkciók használhatók.

A TEXT a megadott felhasználói funkciót adattá változtatja. Próbáljuk ki ezt a már ismert VERS funkcióval.

```
?TO VERS :FOLYO
>PR ( SE :FOLYO [VIZE OLAJOS] )
>PR [FURODJ BENNE ALAJOS]
>END
VERS DEFINED
?
?PR TEXT "VERS
[FOLYO] [PR ( SE :FOLYO [VIZE OLAJOS] )] [PR
[FURODJ BENNE ALAJOS]]
```

A TEXT tehát listaként adja vissza a felhasználói funkciót. A lista minden egyes eleme megfelel a felhasználói funkció egy utasításssorának, amely ismét listát képez. Az első elem mindig olyan lista, amely tartalmazza az első sorban használt funkcióparamétereket. Ha a funkciónak nincsenek bemeneti változói, akkor az első lista üres.

Rendeljük a "FOLYO globális változóhoz a TEXT "VERS eredményét. Ezután nyomtassuk ki egymás alá a lista elemeit.

```
?
?TO ALA :LISTA
>IF EMPTYP :LISTA [STOP]
>PR FIRST :LISTA
>ALA BF :LISTA
>END
ALA DEFINED
?
?MAKE "FOLYO TEXT "VERS
?
?ALA :FOLYO
FOLYO
PR ( SE :FOLYO [VIZE OLAJOS] )
PR [FURODJ BENNE ALAJOS]
?
?
```

Természetesen a listaelemek közül bármelyiket kiválaszthatjuk és kinyomtathatjuk. Ennek érdekében iktassunk be még egy utasítássort az ALA-ba.

```
?
TO ALA :LISTA
IF EMPTYP :LISTA [STOP]
TEST LISTP FIRST :LISTA
IFT [ALA FIRST :LISTA]
IFF [PR FIRST :LISTA]
ALA BF :LISTA
END

?
?ALA :FOLYO
FOLYO
PR
(
SE
:FOLYO
VIZE
OLAJOS
)
PR
FURODJ
BENNE
ALAJOS
?
```

Az előző néhány példa már nyilvánvalóvá teszi, hogy minden egyes felhasználói program a legkisebb részletekig menően megvizsgálható és átalakítható, mivel a fordított műveletre is mód van, azaz programadatokból álló listát funkcióvá alakíthatunk. Ez az utasítás a DEFINE. Készítsünk egy "DUNA nevű felhasználói funkciót, amely a :FOLYO globális változó utasításait tartalmazza.

```
?
?DEFINE "DUNA :FOLYO
?
?POTS
TO DUNA :FOLYO
TO ALA :LISTA
TO VERS :FOLYO
```

Ismerkedjünk meg egy kicsit közelebbről a DEFINE-nal. A DEFINE első bemenete a programnév, a második egy lista, amely listák alakjában a felhasználói program utasításssorait tartalmazza. Ez a lista különböző listaműveletek eredménye lehet. A következő példa azt mutatja, hogy különböző listákból egy közös lista készíthető, amely már egy új program. Az IPOLY példában a két :FOLYO kiegészítő listában BUTFIRST-tel az első elemet el kell távolítanunk, mivel a bemenetváltozó itt hibát okozna.

```
?
?DEFINE "IPOLY (SE TEXT "DUNA BF :FOLYO BF :FOLYO)
?
?PO "IPOLY
TO IPOLY :FOLYO
PR ( SE :FOLYO [VIZE OLAJOS] )
PR [FURODJ BENNE ALAJOS]
PR ( SE :FOLYO [VIZE OLAJOS] )
PR [FURODJ BENNE ALAJOS]
PR ( SE :FOLYO [VIZE OLAJOS] )
PR [FURODJ BENNE ALAJOS]
END
```

4. Példák nagyobb programokra

4.1. Magyar nyelvű Logo

E fejezetben egy saját nyelvrendszert alakítunk ki. Magyar nyelvű utasításokat és műveleteket készítünk, és az új nyelv felhasználásával programokat írunk.

A következő utasításokat és műveleteket magyarítjuk:

Teljes alak	Rövidítés	Jelentés
HA		IF
ISMETELD		REPEAT
NYOMTASD	NY	PRINT
URESE		EMPTYP
ADAT		OUTPUT
ELSO		FIRST
ELSONELKUL	EN	BUTFIRST

1. megoldás: Minden egyes említett utasítás vagy művelet egy felhasználói funkció neve legyen. Ugyanez érvényes a rövidítésekre is, amelyeket szintén új funkciókként definiálunk. A fejlécben feltüntetjük a megfelelő bemeneteket. A funkció tartalma csupán egy sor, a fordítás jelentése szerint. Műveletek esetében a sor elejére OUTPUT-ot kell írni. Először egy műveletet és egy utasítást mutatunk be:

```
?
?TO ELSO :OBJ
>OP FIRST :OBJ
>END
ELSO DEFINED
?
```

```
?
?TO NYOMTASD :OBJ
>PR :OBJ
>END
NYOMTASD DEFINED
?
?
?NYOMTASD ELSO ELSO [EN TE]
E
?
```

Most pedig következzenek a fordítások listái:

```
?
?
?TO HA :OBJ :LISTA
>IF :OBJ :LISTA
>END
HA DEFINED
?
?TO ISMETELD :OBJ :LISTA
>REPEAT :OBJ :LISTA
>END
ISMETELD DEFINED
?
?TO NY :OBJ
>PR :OBJ
>END
NY DEFINED
TO URESE :OBJ
OP EMPTYP :OBJ
END

?
?TO ADAT :OBJ
>OP :OBJ
>END
ADAT DEFINED
?
?TO ELSONELKUL :OBJ
>OP BF :OBJ
>END
ELSONELKUL DEFINED
?
?TO EN :OBJ
>OP BF :OBJ
>END
EN DEFINED
?
?
```

Az EGYMASALA program az új nyelvet teszteli. A beírt szót függőlegesen, egymás alá nyomtassuk ki. Az ALLJ vezérlő szerkezet – mint látható – nem a STOP-ot helyettesíti. Ha ui. STOP-ot használnánk, akkor az csak a HA

műveleten belül hajtódna végre, és nem állítaná le az EGYMASALA-t, ezért a HA-ban TOPLEVEL-t alkalmaztunk.

```
?
?
?
?TO EGYMASALA :SZO
>HA URESE :SZO [ALLJ]
>NY ELSO :SZO
>EGYMASALA EN :SZO
>END
EGYMASALA DEFINED
?
?
?TO ALLJ
>THROW "TOPLEVEL
>END
ALLJ DEFINED
?
?EGYMASALA "MEREDEK
M
E
R
E
D
E
K
```

A magyar nyelvű műveleteket és utasításokat csomaggá összekapcsolva a tárban tarthatjuk. A csomagok kezelését a 3.2. szakaszban már ismertettük.

Ilyen felhasználói funkciók definiálásához hasznos segítséget nyújt a MUVELETEK segédrutin. Itt egy listában megadjuk a Logo műveleteket, majd egy további listában az új, általunk definiált műveletek állnak. Ezt a két listát és a programot lemezen tároljuk. Lássuk a MUVELETEK program listáját és egy rövid tesztet.

```
?
?
?TO MUVELETEK :UJ :REGI
>IF EMPTYP :UJ [STOP]
>DEFINE FIRST :UJ LIST [OBJ] ( LIST "OP FIRST :REGI ":OBJ )
>MUVELETEK BF :UJ BF :REGI
>END
MUVELETEK DEFINED
?
?
?MUVELETEK [UTOLSO UNELKUL] [LAST BL]
?
?MUVELETEK [SZAM UTOLSO] [COUNT LAST]
?
?PO "SZAM PO "UTOLSO
TO SZAM :OBJ
OP COUNT :OBJ
END
```

```
TO UTOLSO :OBJ
OP LAST :OBJ
END

?
```

Az utasításokat megfelelő módon rutinokkal állíthatjuk elő. Ehhez a MUVELETEK 3. sorából eltávolítjuk az "OUTPUT szót és egy új funkcionevet – pl. UTASITASOK-at – írunk be.

```
?
?
?TO UTASITASOK :UJ :REGI
>IF EMPTYP :UJ [STOP]
>DEFINE FIRST :UJ LIST [OBJ] ( LIST FIRST :REGI ":OBJ
>UTASITASOK BF :UJ BF :REGI
>END
UTASITASOK DEFINED
?
```

2. megoldás: Az előbbi megoldás valamennyi Logo rendszerben használható. Egyes Logo rendszerek megengedik, hogy a rendszerfunkciók nevét megváltoztassuk. Ebben az esetben nem kell új funkciókat írnunk. Ilyenkor csak a Logo funkciók átdefiniálásának megfelelő üzemmódot kell beállítanunk. Ez a következő utasítással valósítható meg:

MAKE "REDEFP "TRUE.

Ezután a Logonak csak meg kell mondanunk, hogy mi legyen a funkció új neve. A Logo funkció definíciója már az új néven fog szerepelni. Az utasítás pl. a következő:

COPYDEF "NYOMTASD "PRINT.

COPYDEF után tehát beírjuk az új nevet, majd pedig a Logo funkció nevét.

Mindjárt teszteljük is a Logo funkciók átkeresztelésének ezt a lehetőségét. Kereszteljük át az EGYMASALA példánkban használt Logo funkciókat. Először azonban az ALLJ helyére írjunk STOP-ot.

```
?MAKE "REDEFP "TRUE
?
?COPYDEF "HA "IF
?COPYDEF "URESE "EMPTYP
?COPYDEF "NYOMTASD "PRINT
?COPYDEF "ELSO "FIRST
?COPYDEF "ELSONELKUL "BF
?
TO EGYMASALA :SZO
HA URESE :SZO [STOP]
NY ELSO :SZO
EGYMASALA EN :SZO
END
```

Ha most lefuttatjuk az EGYMASALA-t, akkor mindennek rendben kell lennie. A STOP leállási feltétel megszakítja az EGYMASALA futását, mert a Logo ebben az esetben a HA-t nem felhasználói funkciónak, hanem rendszerfunkciónak tekinti. Ezt ellenőrizhetjük a következő módon:

```
?PO "HA
HA IS A PRIMITIVE
?
```

A Logo funkciók magyarosításához hasznos segédrutin a következő program. A régi és új nevek listáival együtt tároljuk ezt a programot lemezen, így bármikor gyorsan elkészíthetjük a magyar nyelvű Logot.

```
?
?TO ATKERESZTEL :UJ :REGI
>MAKE "REDEFP "TRUE
>IF EMPTYP :UJ [STOP]
>COPYDEF FIRST :UJ FIRST :REGI
>ATKERESZTEL BF :UJ BF :REGI
>END
ATKERESZTEL DEFINED
?
```

Az Apple-Logoban még a TO és GO funkciók is lefordíthatók. A LABEL-lel megadott belépési pont neve azonban nem változtatható meg, mivel gépi szinten az ugrási utasítás után mindig a LABEL jelének kell állnia.

```
?ATKERESZTEL [PROGRAM UGORJ] [TO GO]
?
?PROGRAM "CIKLUS :SZO
>LABEL "KEZDET
>NYOMTASD :SZO
>NYOMTASD [CTRL-G-VEL LEALLITANI]
>UGORJ "KEZDET
>
>END
CIKLUS DEFINED
?
?
?CIKLUS [HAT EZ AZ!]
HAT EZ AZ!
CTRL - G - VEL LEALLITANI
HAT EZ AZ!
CTRL - G - VEL LEALLITANI
HAT EZ AZ!
CTRL - G - VEL LEALLITANI
HAT EZ AZ!
CTRL - G - VEL LEALLITANI
STOPPED! IN CIKLUS:
NYOMTASD [CTRL - G - VEL LEALLITANI]
?
```

4.2. Adatfeldolgozás

A következőkben megismerkedünk az adatfeldolgozás elvével, amely lehetővé teszi, hogy egyszerre több adatot kezeljünk és közvetlenül hozzáférjünk a rekordokhoz. Adatállományon azonos jellegű adatrekordok külső adathordozón (lemezen) levő gyűjteményét értjük. A rekord több adatmezőből áll és bizonyos tényeket tartalmaz, pl. egy diák címét, egy ügyfél adatait stb.

A Logoban nincsenek speciális adatkezelő funkciók, csak a SAVE és a LOAD segítségével áll a lemezekkel kapcsolatban. Igen könnyen előállíthatunk azonban alkalmas adatfeldolgozó rendszereket, hiszen a Logoban nagy felhasználói funkciókat definiálhatunk és használhatunk segédeszközként. Ezek az adatkezelő funkciók rejtve a munkatárban a hasznos utasításkészlethez tartozhatnak. A következőkben egy programvázlat segítségével állítjuk fel az adatfeldolgozás elvét, az adatállományokat és az új utasításokat*:

```
?
?PO "FILEFELDOLGOZAS
TO FILEFELDOLGOZAS
FILENYITAS "FILEX
FILENYITAS "FILEY
KEZDOERTEK.BEALLITAS
LABEL "ISMETELD
REKORDOLVASAS "FILEX :REKORDSZAMX
REKORDOLVASAS "FILEY :REKORDSZAMY
REKORDOK.FELDOLGOZASA
IF FILEVEG [FILEZARAS]
GO "ISMETELD
END

?
```

Mintafájlként képzeljünk el egy egyszerű címállományt, amely a következő adatmezőkből áll:

SORSZAM	NEV	KERESZTNEV	UTCA	IR. SZAM	HELYSEG	TELEFON

* Ez valóban csak vázlatnak tekinthető, mert mint program nem működik, tehát nem a további programok vezérlő programja (Lektor).

Ha a CIMEK állományból a 7. rekordot akarjuk kikeresni, akkor csupán a következő olvasási utasítást kell beírnunk:*

```
?REKORDOLVASAS "CIMEK 7
?PONS
TELEFON IS 99999
HELYSEG IS BUDAPEST
IR.SZAM IS 1089
UT IS VILLAM.15
KERESZTNEV IS LUKACS
VEZ.NEV IS KOVACS
SORSZAM IS 7
@REKORD IS []
?
?
```

Ha egy másik állomány valamelyik rekordját is beolvastatnánk, akkor a munkatárban további globális változók lennének. Az olvasási vagy írási utasítás első alkalmazása előtt minden állományt a FILENYITAS-sal kell megnyitni. Programvázlatunk feldolgozó részében a felhasználói funkciók az adatrekordok ismert adataival módosításokat, ellenőrzéseket vagy nyomtatási műveleteket végeznének, esetleg új adatrekordokat képeznének és azt egy új adatrekordba írnák.

Adatfeldolgozó koncepciónk tehát a következő alaputasításokkal dolgozik:

FILENYITAS	<fájlnév>;	
FILEZARAS	<fájlnév>;	
REKORDOLVASAS	<fájlnév>;	<rekordszám>;
REKORDIRAS	<fájlnév>;	<rekordszám>;

Ezek az állandó alaputasítások lehetővé teszik, hogy egyszerre több fájlhoz hozzáférjünk.

Hogy oldjuk meg ezt a feladatot Logoban?

A PACKAGE utasítás megengedi, hogy a munkatárban a funkciók és/vagy változók bizonyos csoportjait címkékkel lássuk el. Ezzel a megjelölt adatok elkülönítve vihetők lemezre, ill. hozhatók vissza. Az adatrekord egyes mezői tehát egyértelműen jelölhetők az adott állomány nevével. Ily módon egy adatrekord mezőit más állományok rekordjaitól megkülönböztethetjük. Különböző adatrekordok azonos nevű adatmezői is kezelhetők ily módon. Ennek ellenére feltétlenül kerülnünk kell, hogy különböző adatmezők nevei azonosak legyenek.

Minden egyes állomány tulajdonságait egy tulajdonságlistában jegyzi fel a gép. Ez a FILENYITAS-sal történik. A tulajdonságlista neve maga az állo-

* A REKORDOLVASAS funkció három programmal alább található.

mánynév. Ennek egyik fontos tulajdonsága a rekordfelépítés ("REKORDDEF), amely az adatrekordok mezőneveinek listáját tartalmazza. Aki ehhez az adatkezeléshez további megkülönböztető jeleket vagy belépési karaktereket akar hozzáfűzni, az ugyancsak ebbe a tulajdonságlistába írja be, és készítse el a megfelelően megfogalmazott ellenőrző funkciókat.

A lemezen minden rekord egy lemezfájlt képez a fájlnévből és a rekordszámból. Lássuk a CIMEK állományt:*

```
?
DISK VOLUME 254


A 2 HELLO
T 2 CIMEKØ.LOGO
T 2 CIMEK1.LOGO
T 2 CIMEK2.LOGO
T 2 CIMEK3.LOGO
T 2 CIMEK4.LOGO
T 2 CIMEK5.LOGO
T 2 CIMEK6.LOGO
T 2 CIMEK7.LOGO
?
```

A példa kedvéért töltsük be a CIMEKØ-t és a CIMEK1-et a munkatárba:

```
LOAD "CIMEKØ
LOAD "CIMEK1

PONS
@REKORD IS [1 KOVACS LUKACS VILLAM.15 1089 BUDAPEST
99999]
@REKORDDEF IS [SORSZAM VEZ.NEV KERESZTNEV UT IR.SZAM
HELYSEG TELEFON]
?
?
```

Az állomány szerkezetét " REKORDDEF globális változóban találjuk. A CIMEK1 tartalma egy olyan lista, amely az adatmezőket tartalmazza. A lista neve " REKORD. Egyszerű értékadási utasítással ennek tetszőleges nevet és értéket adhatunk. Lássuk a megfelelően definiált adatkezelő utasításokat:

```
?
?TO FILENYITAS :FILE
>LOAD WORD :FILE "Ø
>PPROP :FILE "@REKORDDEF :@REKORDDEF
>PPROP :FILE "VALPKG :FILE
>END
FILENYITAS DEFINED
?
```

* Egy lemez tartalmát a CATALOG utasításra kapjuk meg.

```
?
?TO REKORDOLVASAS :FILE :REKORDSZAM
>LOAD (WORD :FILE :REKORDSZAM)
>PPROP :FILE "@REKORD :REKORD
>ERTEKADAS GPROP :FILE "@REKORDDEF
>END
REKORDOLVASAS DEFINED
?
?
?TO ERTEKADAS :@MEZO
>REPEAT COUNT :@MEZO [MAKE FIRST :@MEZO FIRST
:@REKORD MAKE "@REKORD BF :@REKORD MAKE "@MEZO BF
:@MEZO]
>END
ERTEKADAS DEFINED
```

A FILENYITAS nyitófunkció beolvassa a kívánt állomány ismérveit és ezeket az ismérveket elhelyezi a :FILE tulajdonságlistába, amelynek csomagneve ugyancsak :FILE.

```
?FILENYITAS "CIMEK

?PONS
@REKORDDEF IS [SORSZAM VEZ.NEV KERESZTNEV UT
IR.SZAM HELYSEG TELEFON]

?PR PLIST "CIMEK
VALPKG CIMEK @REKORD [7 KOVACS LUKACS VILLAM.15
1089 BUDAPEST 99999]
@REKORDDEF [SORSZAM VEZ.NEV KERESZTNEV UT IR.SZAM
HELYSEG TELEFON]
?
?
```

Olvassunk be egy adatrekordot:

```
?REKORDOLVASAS "CIMEK 1

?PR PLIST "CIMEK
@REKORD [1 KEK VIRAG LILIOM.10 1000 BP 123456]
VALPKG CIMEK @REKORDDEF [SORSZAM VEZ.NEV KERESZTNEV
UT IR.SZAM HELYSEG TELEFON]

?PONS
TELEFON IS 123456
HELYSEG IS BP
IR.SZAM IS 1000
UT IS LILIOM.10
KERESZTNEV IS VIRAG
VEZ.NEV IS KEK
SORSZAM IS 1
@REKORD IS []
@REKORDDEF IS [SORSZAM VEZ.NEV KERESZTNEV UT
IR.SZAM HELYSEG TELEFON]
?
?
```

Adatállományunk tulajdonságlistájába még a REKORD-ban levő adatrekordok tartalma kerül be. Az ERTEKADAS funkció előállítja a megfelelő mezőneveket és az aktuális értékeket. Az utolsó adatrekordnak a :FILE *tulajdonságlistában* való megőrzése csak biztonsági célokat szolgál, mivel az értékadás után a REKORD adatrekord üres, és a későbbi feldolgozás során az ezzel kapcsolatos nehézségek emiatt elmaradnak.

Az adatrekordok lemezre írása hasonló alapelv szerint történik. A funkció az adatmező tartalmát listává állítja össze, amely megfelelő módon címkeként kapja meg az "IRAS szót. Ezt az "IRAS nevű csomagot a tárból a kívánt fájlnévvel lemezre írjuk

```
?
?
?TO REKORDIRAS :FILE :REKORDSZAM
>REKORDKEPZES GPROP :FILE "@REKORDDEF
>PPROP :FILE "@REKORD :@REKORD
>PPROP "@REKORD "VALPKG "IRAS
>CATCH "ERROR [ERASEFILE WORD :FILE :REKORDSZAM]
>
>SAVE WORD :FILE :REKORDSZAM "IRAS
>END
REKORDIRAS DEFINED
?
?
?
?TO REKORDKEPZES :@MEZO
>MAKE "@REKORD []
>REPEAT COUNT :@MEZO [MAKE "@REKORD SE :@REKORD
>THING FIRST :@MEZO MAKE "@MEZO BF :@MEZO]
>END
REKORDKEPZES DEFINED
?
```

Mielőtt ezzel a rutinnal dolgoznánk, közvetlen üzemmódban fájlleírásként az adatrekord felépítést rögzítenünk kell és lemezre kell írnunk. Előtte mindent törölnünk kell vagy el kell rejtenünk, még a tulajdonságlistákat is! Csak a " REKORDDEF változó legyen a lemezen levő fájlleírás tartalma.

```
?MAKE "@REKORDDEF [SORSZAM VEZ.NEV KERESZTNEV UT
IR.SZAM HELYSEG TELEFON]
?SAVE "CIMEKØ
Ø PROCEDURES SAVED

CATALOG

DISK VOLUME 254

 A    2 HELLO
 T    2 CIMEKØ.LOGO
?
```

Végül bemutatunk egy elemi adatrögzítő programot, amely ellenőrző- és hibarutinokat nem tartalmaz, de ezekkel könnyen kiegészíthető. A meglevő rekordok átírása (a rekordszám hibás beírása) elleni egyszerű védelem, hogy elhagyjuk a CATCH "ERROR... sort. Hiba esetén a program leáll. Legalábbis az első adatrögzítésre ez így alkalmas. Módosításkor a sornak megint rendelkezésre kell állnia. A módosítás a mezőnév lekérdezésével vagy egy menünek megfelelő módosítással érhető el. Az új érték lesz ezután az adatmező tartalma. Végül csak a REKORDIRAS rutint hívjuk, és a módosított rekord a lemezen is van.

```
TO ROGZITES :FILE
FILENYITAS :FILE
LABEL "START
DE.REKORD :@REKORDDEF
REKORDIRAS :FILE FIRST THING FIRST :@REKORDDEF
IF NOT VEGE [GO "START]
FILEZARAS :FILE
END
TO DE.REKORD :LISTA
CLEARTEXT
SETCURSOR [0 0]
LZ 2
PR [KEREM BEIRNI A KOVETKEZO ADATOKAT...]
LZ 2
REPEAT COUNT :LISTA [DE.SZO FIRST :LISTA LZ 1 MAKE
"LISTA BF :LISTA]
END
TO DE.SZO :@NEV
TAB 5 CHAR 32
TYPE :@NEV
TAB 16 ".
TYPE ":
MAKE ":@NEV RL
END
TO TAB :POZ :JEL
IF :POZ = FIRST CURSOR [STOP]
TYPE :JEL
TAB :POZ :JEL
END
TO LZ :N
REPEAT :N [PR " ]
END

?
TO VEGE
SETCURSOR [0 23]
TYPE [VANNAK TOVABBI ROGZITENDO REKORDOK?]
TYPE CHAR 7
TEST FIRST RL = "IGEN
IFT [OP "FALSE]
IFF [OP "TRUE]
END
```

```
TO FILEZARAS :FILE
THROW "TOPLEVEL
END

?

?ROGZITES "CIMEK


KEREM BEIRNI A KOVETKEZO ADATOKAT...


     SORSZAM....:1

     VEZ.NEV....:KEK

     KERESZTNEV.:VIRAG

     UT.........:LILIOM.10

     IR.SZAM....:1000

     HELYSEG....:BP

     TELEFON....:123456

0 PROCEDURES SAVED
VANNAK TOVABBI ROGZITENDO REKORDOK?IGEN
?
```

A két következő példa az adatfeldolgozást mutatja be. Az elsőben bizonyos rekordokon levő címeket felragasztható címkékre nyomtatunk:

```
TO KIIR :SORSZ
FILENYITAS "CIMEK
LABEL "START
IF EMPTYP :SORSZ [STOP]
REKORDOLVASAS "CIMEK FIRST :SORSZ
FELDOLGOZNI
MAKE "SORSZ BF :SORSZ
GO "START
END
TO FELDOLGOZNI
PR "  PR "
( PR :VEZ.NEV :KERESZTNEV )
( PR :UT )
( PR :IR.SZAM :HELYSEG )
END

?KIIR [7 1]
```

```
KOVACS LUKACS
VILLAM.15
1089 BUDAPEST
```

```
KEK VIRAG
LILIOM.10
1000 BP
?
?
```

Amennyiben az adatokat sorosan, egy kiindulási számtól egy végszámig akarunk kinyomtatni, akkor a következő általános funkciót használhatjuk, amely minden állományra alkalmazható.

```
TO KIIR2 :TOL :IG :FILE :MI
FILENYITAS :FILE
LOCAL "ANZ
MAKE "ANZ SUM 1 ( :IG - :TOL )
REPEAT :ANZ [REKORDOLVASAS :FILE :TOL RUN :MI MAKE
"TOL :TOL + 1]
END

?KIIR2 1 7 "CIMEK [( PR :VEZ.NEV :TELEFON :HELYSEG )]
KEK 123456 BP
NAGY 345298 PECS
KIS 43252 GYOR
PAL 87654 EGER
STAUB 37491 VESZPREM
KOVACS 99999 BUDAPEST


?KIIR2 3 4 "CIMEK [FELDOLGOZNI]


KIS PETER
ROZSA .43
8040 GYOR


PAL ODON
DOZSA.31
2345 EGER
?
```

A lemez tartalomjegyzékében csak 70 tétel fér el. Ez még bemutató példák esetén is túlságosan kevés adatrekord, különösen akkor, ha egyszerre több adatállománnyal dolgozunk. Ilyenkor több rekordból egy blokkot képzünk, majd a blokkokat sorszámozva tároljuk a lemezen. Így tízesével csoportosítva már több mint 700 adatrekordunk lehet.

A módosítás egyszerű. Csupán a rekordszámokat kell megfelelő módon átszámítanunk. Az 1...10 adatrekord blokkszáma 1, a 11...20 adatrekordé 2 stb. A BLOKKSF funkció megoldja ezt a feladatot:

```
?
?TO BLOKKSF :N
>IF LAST :N = 0 [OP BL :N]
>OP SUM 1 INT :N / 10
>END
BLOKKSF DEFINED
?
?
```

Az adatkezelő utasítások közé most már csak a BLOKKSF utasítást kell megfelelően beilleszteni. Ezután az ERTEKADAS és REKORDKEPZES

funkciókat úgy módosítjuk, hogy több rekordból álló listát (blokkot) készíthessünk. Olvasáskor ITEM-mel és a rekordszám utolsó jegye alapján kiemeljük a rekordot a blokkból. A rekordszám utolsó jegye szabja meg ui., hogy a blokk melyik pozíciójáról kell a rekordot kiemelni, ill. hova kell elhelyezni.

4.3. Függvények rajzolása

Matematika órákon szokásos feladat, hogy valamilyen függvény értéktáblázatát készítik el. Az értéktáblázat összetartozó x és y értékeit a koordinátarendszerben ábrázolják. Ha a koordinátatengelyeken ésszerű léptéket választottunk, akkor egy görbét (a függvény képét) kapjuk. Ennek megoldása a számítógép számára klasszikus feladat.

Nevezzük a programot GRAPH-nak! A program rajzoljon polinomokat. Szemléltetés céljából válasszuk a következő negyedfokú polinomot:

$$y = 6x^4 - 16x^3 + 12x^2.$$

A polinomok általános alakja a következő:

$$y = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_4 x^4 + a_3 x^3 + a_2 x^2 + a_1 x^1 + a_0.$$

Példánkban a negyedfokú polinom együtthatóértékei a következők:

$a_4 = 6;$
$a_3 = -16;$
$a_2 = 12;$
$a_1 = 0;$
$a_0 = 0.$

Rajzoltassuk ki a számítógéppel ezt a görbét.

```
?GRAPH [6 -16 12 0 0] -2. 2. .2 []
       -2.          272.

      -1.8     195.178
      -1.6     135.578                          *
      -1.4     90.4737                  *
      -1.2     57.3697          *
       -1.     34.0001       *
  -.800001     18.3297     *
  -.600001     8.55365    *
  -.400001     3.09763 *
  -.200001     .617609 *
 -1.3113N6 2.06342N11 *
   .199999     .361596 *
   .399999     1.04959 *
   .599999      1.6416 *
   .799998      1.9456 *
   .999998          2. *
       1.2      2.0736 *
       1.4     2.66559 *
       1.6     4.50557 *
       1.8     8.55355   *
        2.     15.9999    *
```

Az x és y értékeket felülről lefelé két oszlopban jobbra ütköztetve egymás alá írjuk. Minden egyes sor x értéke gyakorlatilag az x tengelyt jelképezi. A sor fennmaradó részében lehet kirajzolni az y értéket. Függőleges tengelyt nem rajzolunk. Ezt bárki szükség esetén vonalzóval és ceruzával könnyen megrajzolhatja.

Az x értékek -2-től 0,2 egységenként nőnek $+2$-ig. Figyeljük meg a GRAPH hívását. Lista alakjában ismét találkozunk az együtthatókkal, ezt követi az x értéktartománya és az x növekménye. Utolsó helyen egy üres lista áll, ezzel zárjuk le a bemenetet.

Először a szükséges funkciókat mutatjuk be. Tekintettel arra, hogy a POTS fordított sorrendben írja ki a funkciókat, visszafelé kell olvasni*.

```
?
?POTS
TO COUNTW :SZO
TO SZOBOLLISTA :SZO
TO TOLTO :JEL :SZAM
TO JOBBRA :MI :HOSSZ :JEL
TO WHILE :FELTETEL :TEDD
TO FOR :VAR@ :A@ :E@ :S@ :UTASITAS
TO NYOMTATO.BE
TO START
```

* COUNTW, SZOBOLLISTA és TOLTO a 9. fejezetben található.

```
TO TAB :@
TO KEPPONT :@
TO SZAMPAR
TO NYOMTAT.Y.ERTEK :Y
TO HORNER :POLY :ARG
TO NYOMTAT
TO MINMAX :MIN :MAX :YW
TO KEPHATAROK :MINMAX
TO YERTEKEK :FVX :XA :XE :DX
TO GRAPH :FVX :XA :XE :DX :KORLATOK
?
?
```

A GRAPH főprogram:

```
?
?TO GRAPH :FVX :XA :XE :DX :KORLATOK
>YERTEKEK :FVX :XA :XE :DX
>TEST EMPTYP :KORLATOK
>IFT [KEPHATAROK MINMAX FIRST :YW FIRST :YW :YW]
>IFF [KEPHATAROK :KORLATOK]
>
>WHILE [NOT EMPTYP :YW] [NYOMTAT]
>END
GRAPH DEFINED
?
```

Az YERTEKEK funkció az értéktáblázat y koordinátáit határozza meg. Az YERTEKEK eredménye az : YW változóban van, lista formájában. A KEPHATAROK számítja ki az y értékek kirajzolási léptékét. Mivel nyomtatósorban vagy a képernyősorban csak rögzített számú írási pozíció van, valamennyi y értéket ebben a tartományban kell elhelyezni. A legkisebb és a legnagyobb y értéknek éppen a képernyő vagy a nyomtató bal, ill. jobb szélén kell lennie. Az y tengelylépték kiszámítását a KEPHATAROK funkció végzi. A MINMAX a KEPHATAROK számára kiszámítja a legkisebb és a legnagyobb y értéket. A NYOMTAT választás szerint a koordinátákat és a görbét vagy csak a görbét írja ki, az utóbbi esetben a megtakarított terület is a rajz rendelkezésére áll, így a függvény képe pontosabb lesz.

Lássuk most minden részletében a GRAPH programot.

```
?
?TO YERTEKEK :FVX :XA :XE :DX
>MAKE "YW []
>FOR "I :XA :XE :DX [MAKE "YW SE :YW HORNER :FVX :I]
>END
YERTEKEK DEFINED
?
?
?TO KEPHATAROK :MINMAX
>MAKE "MIN FIRST :MINMAX
```

```
>MAKE "MAX LAST :MINMAX
>TEST :MIN<Ø
>IFT [MAKE "LEPES BF (:TARTOMANY / (:MIN - :MAX))]
>IFF [MAKE "LEPES (:TARTOMANY / (:MAX - :MIN))]
>END
KEPHATAROK DEFINED
?
?
?TO MINMAX :MIN :MAX :YW
>IF EMPTYP :YW [OP SE :MIN :MAX]
>IF FIRST :YW < :MIN [OP MINMAX FIRST :YW :MAX BF :YW]
>TEST FIRST :YW > :MAX
>IFT [OP MINMAX :MIN FIRST :YW BF :YW]
>IFF [OP MINMAX :MIN :MAX BF :YW]
>END
MINMAX DEFINED
?
?
?TO NYOMTAT
>TEST :KOORDINATAK
>IFT [TYPE SZAMPAR]
>NYOMTAT.Y.ERTEK FIRST :YW
>MAKE "XA :XA + :DX
>MAKE "YW BF :YW
>END
NYOMTAT DEFINED
?
?
?TO HORNER :POLY :ARG
>LOCAL "S MAKE "S Ø
>LABEL "START
>IF (COUNT :POLY) = 1 [OP SUM :S FIRST :POLY]
>MAKE "S (SUM :S FIRST :POLY) * :ARG
>MAKE "POLY BF :POLY
>GO "START
>END
HORNER DEFINED
?
?
?TO NYOMTAT.Y.ERTEK :Y
>TEST OR :Y < :MIN :Y > :MAX
>IFT [PR "\  STOP]
>TEST :Y < Ø
>IFT [TAB KEPPONT :Y - :MIN PR "\  * STOP]
>TEST :MIN < Ø
>IFT [TAB KEPPONT :Y + BF :MIN PR "\  * STOP]
>IFF [TAB KEPPONT :Y - :MIN PR "\  *]
>END
NYOMTAT.Y.ERTEK DEFINED
?
?
?TO SZAMPAR
>OP SE JOBBRA :XA 10 CHAR 32 JOBBRA FIRST :YW 10
CHAR 32
>END
SZAMPAR DEFINED
?
```

```
?
?PO "KEPPONT
TO KEPPONT :@
OP ROUND (:@ *
 :LEPES)
END

?
?TO TAB :@
>IF NOT (:@ > 1) [STOP]
>REPEAT (:@ - 1) [TYPE "\  ]
>END
TAB DEFINED
?
?
?PO "JOBBRA
TO JOBBRA :MI :HOSSZ :JEL
OP WORD TOLTO :JEL :HOSSZ - COUNTW :MI :MI
END


TO START
PR [KELL NYOMTATO? ( I / N )]
TEST EQUALP FIRST RL "I
IFT [MAKE "NYOMTATO "TRUE]
IFF [MAKE "NYOMTATO "FALSE]
PR [A KOORDINATAKAT KIIRJUK? ( I / N )]
TEST EQUALP FIRST RL "I
IFT [MAKE "KOORDINATAK "TRUE]
IFF [MAKE "KOORDINATAK "FALSE]
TEST :KOORDINATAK
IFT [IF :NYOMTATO [MAKE "TARTOMANY 37 NYOMTATO.BE]
[MAKE "TARTOMANY 17]]
IFF [IF :NYOMTATO [MAKE "TARTOMANY 60] [MAKE
"TARTOMANY 37]]
END

 TO NYOMTATO.BE
 .PRINTER 1
 PR ( WORD CHAR 27 CHAR 81 CHAR 14 )
 PR WORD CHAR 9 "60N
 END
```

Az YERTEK csupán egy FOR ciklusból és abból az utasításból áll, amely megadja, miképpen kell kiszámítani az y értéket. A számítási előírás a HORNER. A polinomok y értékeit ui. a Horner-elrendezéssel számítjuk ki.

A képhatárokon a leképezési tartományt egy y értékegységre kiszámítjuk és a :LEPES-ben tároljuk. Tegyük fel, hogy az y értékek 1 és 100 közé esnek, amelyet a képernyő 38 pozíciójának kell megfeleltetnünk. A képernyőn levő 1. pozíció megfelel az y = 1-nek a 38. pozíció a 100-nak. Egy-egy y értékre tehát csak 0,38 pozíció van. Ebből következik, hogy egy írási pozíción több átszámított y értéknek kell osztozkodnia. Az átszámítandó tartomány a :TARTOMANY változóban van, ami a nyomtatótól és a képernyőtől függően változik. A GRAPH használata előtt tehát ennek az értékét be kell állítanunk.

Ha a teljes képernyősort kihasználjuk, akkor a következőt kell beírnunk: MAKE "TARTOMANY 38. A START indítóprogram az értékeket párbeszédes formában kérdezi le. Itt dönthető el az is, hogy a görbét koordinátaértékekkel vagy anélkül akarjuk kirajzolni. A :KOORDINATAK? változóba "TRUE vagy "FALSE kerül. A :KOORDINATAK?-ra a NYOMTAT rutinban van szükség. A START futása után a programot úgy indítjuk, mint a bevezető példában, azaz legépeljük a GRAPH-ot és a hozzá tartozó bemeneti értékeket.

```
?
?START
KELL NYOMTATO? ( I / N )
I
A KOORDINATAKAT KIIRJUK? ( I / N )
I
```

Előző függvényábránk mutatja, hogy a –0,4...1,4 x tartományban az y értékek egybeesnek. Nyomtassuk ki a görbét még egyszer, most azonban csak ebben a tartományban, ahol az :X lépéshosszt is kisebbre vettük.

```
?GRAPH [6 -16 12 0 0] -.3 1.4 .05 []
       -.3      1.5606                          *
      -.25     1.02344                  *
       -.2       .6176            *
      -.15     .327038       *
       -.1       .1366     *
-5.00001N2  3.20376N2 *
-5.96046N8 4.26326N14 *
 4.99999N2  2.80374N2 *
 9.99999N2       .1046   *
       .15     .219037      *
        .2       .3616        *
       .25     .523437          *
        .3       .6966             *
       .35     .874037                *
        .4      1.0496                  *
       .45     1.21804                     *
        .5       1.375                       *
       .55     1.51704                          *
        .6      1.6416                           *
   .649999     1.74704                             *
   .699999      1.8326                              *
   .749999     1.89844                               *
   .799999      1.9456                                *
   .849999     1.97604                                 *
   .899999      1.9926                                 *
   .949999     1.99904                                 *
   .999999          2.                                 *
      1.05     2.00104                                 *
       1.1      2.0086                                 *
      1.15     2.03004                                 *
       1.2      2.0736                                  *
      1.25     2.14843                                   *
       1.3      2.2646                                     *
      1.35     2.43303                                       *
```

Ez az eredmény már szebb. Tehát a szélső értékeknek megfelelően érdemes széthúzni az ábrát.

A GRAPH program csekély módosítással más függvénytípusokra is használható. Írassuk ki a szinuszfüggvényt a 0°...360° tartományban:

```
?GRAPH [SIN :I] 0. 360. 10. []
        0.          0.                    *
       10.     .173648                       *
       20.      .34202                          *
       30.          .5                             *
       40.     .642788                                *
       50.     .766044                                  *
       60.     .866025                                    *
       70.     .939693                                     *
       80.     .984808                                      *
       90.          1.                                      *
      100.     .984808                                      *
      110.     .939693                                     *
      120.     .866025                                    *
      130.     .766044                                  *
      140.     .642788                                *
      150.          .5                             *
      160.      .34202                          *
      170.     .173648                       *
      180.          0.                    *
      190.    -.173648               *
      200.     -.34202            *
      210.         -.5         *
      220.    -.642788       *
      230.    -.766044    *
      240.    -.866025  *
      250.    -.939693 *
      260.    -.984808 *
      270.         -1. *
      280.    -.984808 *
      290.    -.939693 *
      300.    -.866025  *
      310.    -.766044    *
      320.    -.642788       *
      330.         -.5         *
      340.     -.34202            *
      350.    -.173648               *
      360.          0.                    *
```

Az alapfüggvényeken kívül összetett trigonometriai függvények is kirajzolhatók. Rajzoljuk ki pl. az $y = \sin x \cos x$ függvényt!

```
?GRAPH [PRODUCT SIN :I COS :I] 0. 360. 10. []
        0.          0.                   *
       10.      .17101                          *
       20.     .321394                                *
       30.     .433013                                    *
       40.     .492404                                      *
       50.     .492404                                      *
       60.     .433013                                    *
       70.     .321394                                *
       80.      .17101                          *
       90.          0.                   *
      100.     -.17101            *
      110.    -.321394      *
      120.    -.433013  *
      130.    -.492404 *
      140.    -.492404 *
      150.    -.433013  *
      160.    -.321394      *
      170.     -.17101            *
      180.          0.                   *
      190.      .17101                          *
      200.     .321394                                *
      210.     .433013                                    *
      220.     .492404                                      *
      230.     .492404                                      *
      240.     .433013                                    *
      250.     .321394                                *
      260.      .17101                          *
      270.          0.                   *
      280.     -.17101            *
      290.    -.321394      *
      300.    -.433013  *.
      310.    -.492404 *
      320.    -.492404 *
      330.    -.433013  *
      340.    -.321394      *
      350.     -.17101            *
      360.          0.                   *
```

Csak a felhasználói funkciót kell módosítani. A függvény megadásakor ügyelnünk kell arra, hogy a kifejezés függ a :I változótól. Hasonlítsuk össze a GRAPH hívását a két utóbbi példában!

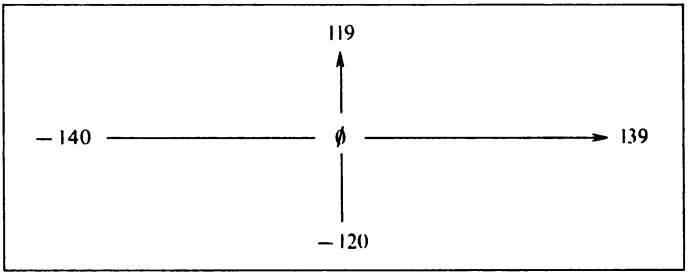
A nyomtatót bekapcsoló NYOMTATO.BE természetesen függ az adott nyomtatótípustól.

A Logo segítségével másképp is megoldható a függvényábrázolás Teknőcgrafikával. A Teknőc pontról pontra kövesse az x, y értékpárokat.

Mit kell ehhez tennünk?

Először is az x tengely és az y tengely léptékeit át kell számítanunk a grafikai tartományokra. Továbbá a SETPOS grafikai utasításnak minden értékpárt lista alakjában kell megadnunk. A Teknőc eredeti helyéről az új helyzet-

be mozdul el. További előnye ennek, hogy koordináta-keresztet is rajzolhatunk. A Teknőc képernyőtartománya a következő:



Csak a következő változtatásokra van szükség:

GRAPH
NYOMTAT
NYOMTAT.Y.ERTEK
START

A START funkciót újraírtuk, most már néhány útmutatást ad a felhasználónak. Az *x* legnagyobb abszolút értékétől függően kiszámítja az *x* érték rajzolási léptékét.

Új funkció a KEZDES törli a grafikai képernyőt és felrajzolja a koordinátakeresztet. X átszámítja az X koordinátát a rajzolási koordinátává, tehát az Y értékeket átszámító KEPPONT megfelelője.

```
TO START
TEXTSCREEN
CLEARTEXT
PR [TEKNOS UZEMMODBAN VAGYUNK!]
PR ""
PR [AZ X - TARTOMANY: -140 - TOL 139 - IG]
PR [AZ Y - TARTOMANY: -120 - TOL 119 - IG]
PR ""
PR [AZ Y - ERTEKEKET AZ Y - TARTOMANYRA ATSZAMOLOM]
PR "  PR ""
PR [AZ X - SKALA LEPESKOZEIT MEG KELL HATAROZNUNK:]
PR [EZ X - MAX ABSZOLUT ERTEKETOL FUGG]
PR "  PR ""
PR [MENNYI LEGYEN X - MAX?]
MAKE "XSKALA FIRST RL
MAKE "XSKALA 140 / :XSKALA
PR "  PR [ES MOST HIVJA A GRAPH PROGRAMOT!]
RUN RL
END
```

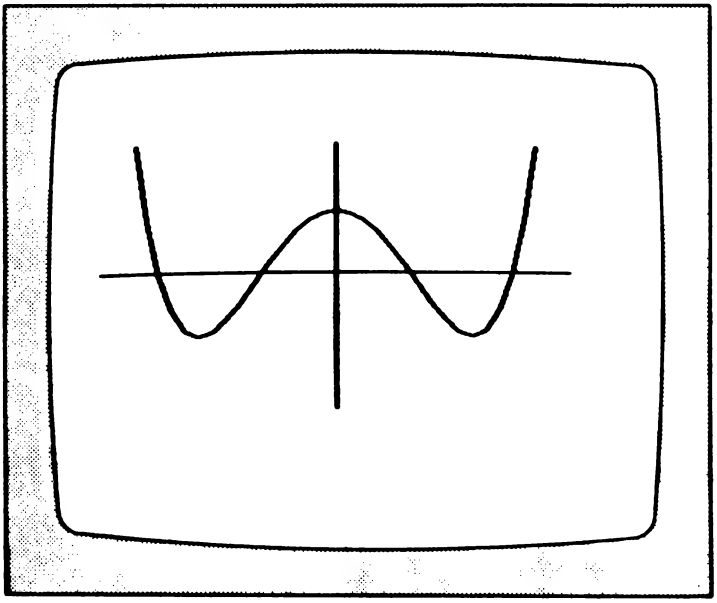
```
?
?
TO GRAPH :FVX :XA :XE :DX :KORLATOK
YERTEKEK :FVX :XA :XE :DX
TEST EMPTYP :KORLATOK
IFT [KEPHATAROK MINMAX FIRST :YW FIRST :YW :YW]
IFF [KEPHATAROK :KORLATOK]
CLEARSCREEN
KEZDES
WHILE [NOT EMPTYP :YW] [NYOMTAT]
END

?
?
TO KEZDES
SETPOS [0 240] HOME
SETPOS [300 0] HOME
FULLSCREEN
PU SETPOS SE X :XA KEPPONT FIRST :YW PD
END

?
?
TO NYOMTAT
NYOMTAT.Y.ERTEK FIRST :YW
MAKE "XA :XA + :DX
MAKE "YW BF :YW
END

?
?
TO NYOMTAT.Y.ERTEK :Y
TEST OR :Y < :MIN :Y > :MAX
IFT [PR "  STOP]
SETPOS SE X :XA KEPPONT :Y
END

?
?
TO X :@
OP :XSKALA * :XA
END
```

A következőkben három függvényt fogunk ábrázolni. Az első függvényképet már ismerjük.*

```
?
?
?
?
?START
TEKNOS UZEMMODBAN VAGYUNK!
```

* A függvényt még egy alkalmas konstanssal meg kell szorozni, hogy az ábráknak megfelelő kép jelenjen meg a képernyőn (Lektor).

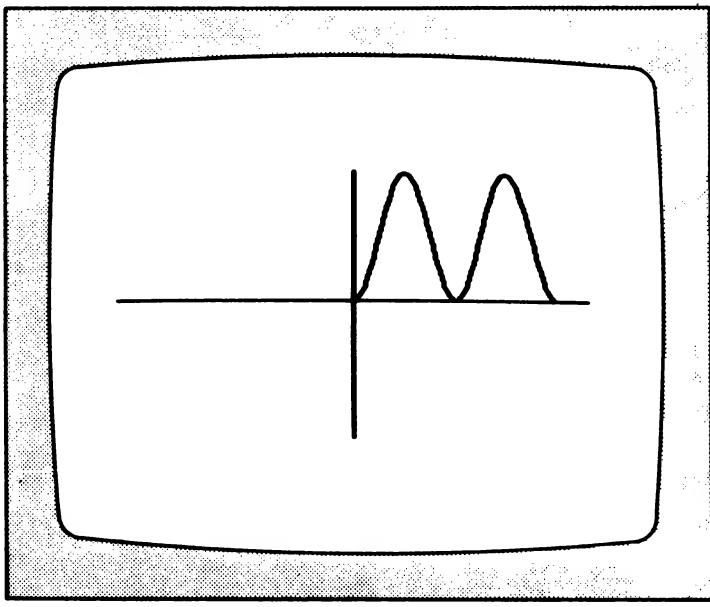
```
AZ X - TARTOMANY: -140 - TOL 139 - IG
AZ Y - TARTOMANY: -120 - TOL 119 - IG

AZ Y - ERTEKEKET AZ Y - TARTOMANYRA ATSZAMOLOM


AZ X - SKALA LEPESKOZEIT MEG KELL HATAROZNUNK:
EZ X - MAX ABSZOLUT ERTEKETOL FUGG


MENNYI LEGYEN X - MAX?
3.

ES MOST HIVJA A GRAPH PROGRAMOT!
GRAPH [.25 0 -2 0 2] -3. 3. .2 []
```



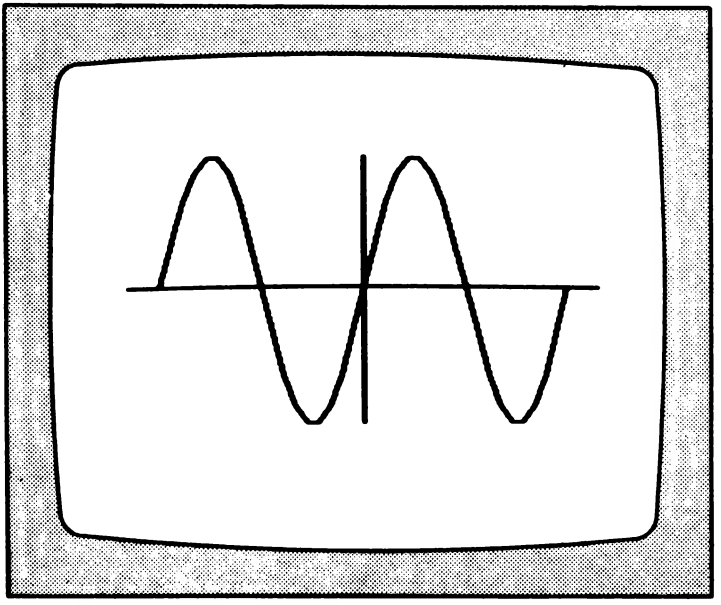
A következő függvény legyen az $y = (\sin x)^2$. Csak a 0...360° közötti értékeket rajzoljuk ki 5°-os lépéshosszal.

A hívás alakja a következő:

GRAPH [PRODUCT SIN :I SIN :I] 0 360 5 [ ].



A következő függvény az $y = \sin x \cos x$ a $-180°...180°$ tartományban, $10°$-os lépéshosszal.



4.4. Az indexes (tömb-) változók használata

Indexes változón (tömbön) olyan adattípust értünk, amely egyidejűleg több érték tárolására alkalmas, az egyes értékeket a változónév és az index együttes használatával lehet elérni. Az ilyen egy- vagy kétdimenziós indexes változókat a matematikában vektoroknak, ill. mátrixoknak nevezik. A kétdimenziós tömb magyarán a táblázat. A táblázat sorokból és oszlopokból áll. A táblázat minden egyes mezőjét meghatározza a sorszám és az oszlopszám. Az egydimenziós tömb a táblázat különleges esete, egy lista, amely pl. csak egy sorból áll, és megfelelő számú elemet (oszlopot) tartalmaz. A Logo lista tulajdonképpen egy tömb, amelyben a lista kívánt elemét nem egy indexszel érhetjük el, hanem a lista elemeit balról vagy jobbról kezdve egyenként dolgozzuk fel. A személyi számítógépek Logo változatai nem ismernek tömbfunkciókat. Ez azonban nem baj, mivel a szükséges elemi funkciókat könnyen magunk is megírhatjuk. Ebben a fejezetben az ilyen alapfunkciókat és ezek

kezelését mutatjuk be. A funkciókat ezután rövid ismertetéssel együtt kilistázzuk. Ezután következnek a mintapéldák.

4.4.1. Felhasználó által definiált táblázatfunkció

A táblázatok definiálására és feldolgozására szolgáló elemi funkciók a következők lehetnek:

- Táblázat definiálása.
- Egy érték kiolvasása a táblázatból.
- Egy érték beírása a táblázatba.
- A definiált táblázat méretének megállapítása.

Ezeken az elemi funkciókon kívül még két segédfunkcióra is szükség van, amelyek segítségével a táblázat valamennyi eleme értékekkel tölthető fel, ill. kinyomtatható.

Tehát a következő alapfunkciókat definiáljuk:

- DEFV (vektor definiálása).
- OV (vektor olvasása).
- IV (vektormező írása).
- DIMV (már definiált vektor dimenziója).

Ismerkedjünk meg az említett funkciókkal egy példa alapján. Definiáljunk egy olyan táblázatot, amely négy sorból és öt oszlopból áll. (A táblázat neve legyen TABLAZAT.)

Milyenek az egyes funkciók?

A táblázat minden egyes sora egy lista. A sor neve az előre megadott névből, valamint a sor számából áll. Tehát a lista minden egyes eleme a táblázat egy oszlopeleme.

```
?
?TO DEFV :NAME :Z :S
>FOR "I 1 :Z 1 [MAKE ( WORD "V :NAME :I) OSZLOP :S]
>MAKE (WORD "V :NAME) SE :Z :S
>END
DEFV DEFINED
?
?
?TO OSZLOP :S
>LOCAL "SP MAKE "SP []
>REPEAT :S [MAKE "SP SE :SP ".]
>OP :SP
>END
OSZLOP DEFINED
?

?DEFV "TABLAZAT 4 5
```

```
?PONS
VTABLAZAT IS [4 5]
VTABLAZAT4 IS [. . . . .]
VTABLAZAT3 IS [. . . . .]
VTABLAZAT2 IS [. . . . .]
VTABLAZAT1 IS [. . . . .]
?
?
?
```

E szerint a megoldás szerint valamennyi változó globális változó. Valamennyi változónévhez hozzáfűztük a V betűt, így a változónevek rögtön felismerhetők és más változókkal nem lehet őket összecserélni. Az OSZLOP funkció a táblázat definiálásakor előállítja az egyes oszlopelemeket és a sor egyes mezőit az egyezményes "üres" jellel látja el. Csak a ". jelet kell más jelre kicserélnünk, és máris másféle az "URES" jel.

A DIMV olyan művelet, amely egy kételemű listát ad eredményül. Az első szám a sorok száma, a második szám a soronkénti oszlopok száma. A DIMV bemenete a táblázat neve.

```
?
?PO "OV
TO OV :NEV :Z :S
CATCH "ERROR [OP ITEM :S THING ( WORD "V :NEV :Z )]
HIBA
END

?
?
?TO HIBA
>LOCAL "HIBA MAKE "HIBA ERROR
>IF EMPTYP :HIBA [STOP]
>PR [HIBAS INDEX?]
>PR :HIBA
>THROW "TOPLEVEL
>END
HIBA DEFINED
?
?
?TO DIMV :NEV
>OP THING WORD "V :NEV
>END
DIMV DEFINED
?

?PR DIMV "TABLAZAT
4 5
?
```

Ha egy táblázatból értéket akarunk olvasni, (OV) akkor a következő három adatra van szükségünk: a tömb nevére, a sorszámra és az oszlopszámra. Itt a mezők tetszőleges adatokkal tölthetők fel – számokkal, szavakkal, sőt

listákkal is. Más programnyelvekben egy tömbön belül rendszerint csak egyféle adattípus lehet. Ráadásul a táblázat dimenziója a program futása közben is módosítható! Számos más nyelvben a táblázat dimenzióját a program elején kell rögzíteni.

A táblázatból valamely elemet úgy olvashatunk ki, hogy a beírt táblázatnévből és a megadott sorszámból megalkotjuk a megfelelő globális változó nevét. Az ismert ITEM funkcióval a lista megfelelő *n*-edik elemét az oszlopszám alapján kiválaszthatjuk. Mivel olvasási és írási műveletek során gyakran előfordul indextúllépés, tanácsos CATCH segítségével ellenőrzött hibakimenetet előkészíteni, hogy éppen a tesztelési időszakban egy hibakezelő funkció kinyomtassa számunkra a lokális változókat. Az itt bemutatott "HIBA funkciót mindenki az igényeinek megfelelően módosíthatja. Ha pl. elhagyjuk a THROW " TOPLEVEL-t, akkor a program a hiba ellenére tovább fut.

```
?
?TO IV :NEV :Z :S :ERTEKV
>CATCH "ERROR [MAKE (WORD "V :NEV :Z) BEILLESZT
>THING (WORD "V :NEV :Z) :S :ERTEKV]
>HIBA
>END
IV DEFINED
?
?
?TO BEILLESZT :V :S :ERTEK
>IF :S = 1 [OP SE :ERTEK BF :V]
>OP SE FIRST :V BEILLESZT BF :V :S - 1 :ERTEK
>END
BEILLESZT DEFINED
?
?DEFV "TABLAZAT 4 5
?
?IV "TABLAZAT 1 1 123
?IV "TABLAZAT 1 2 "DE
?IV "TABLAZAT 1 3 "ABC
?IV "TABLAZAT 1 4 "HALLO
?IV "TABLAZAT 1 5 "HUHU
?
?IV "TABLAZAT 2 2 34
?IV "TABLAZAT 3 3 "LALI
?IV "TABLAZAT 4 4 "ICA
?IV "TABLAZAT 5 5 12.3
```

Ha egy táblázatba az IV funkcióval értékeket írunk, akkor négy bemenetet kell megadnunk: a tömb nevét, sorszámát, oszlopszámát és a beírandó szót.

Tehát az írófunkció alapelvében megegyezik az olvasó funkcióval. A BEILLESZT művelet végzi el a fő munkát. A kívánt mezőelem előtt álló valamennyi sorelemet végigjárjuk. Ezután a listát kiegészítjük az új elemmel.

A régi táblázatsor fennmaradó részét az első helyen álló elem kivételével hozzácsatoljuk. Itt is megvan a hibakimenet.

```
?TO NYOMV :NEV
>FOR "I 1 FIRST DIMV :NEV 1 [FOR "J 1 LAST DIMV
>:NEV 1 [TYPE LV :NEV :I :J TYPE "\  ]PR "\  ]
>END
NYOMV DEFINED
?
?
?TO TOLTV :NEV
>FOR "I 1 FIRST DIMV :NEV 1 [FOR "J 1 LAST DIMV
>:NEV 1 [SV :NEV :I :J ERTEKV]]
>END
TOLTV DEFINED
?
?
?
TO FOR :VAR@ :A@ :E@ :S@ :UTASITAS
LOCAL :VAR@
MAKE :VAR@ :A@
REPEAT SUM 1 (:E@ - :A@) / :S@
[RUN :UTASITAS MAKE :VAR@ :S@ + THING :VAR@]
END

?
?
?TO ERTEKV
>TYPE (SE :NEV :I :J [MEZOVEL] "...:)
>OP FIRST RL
>END
ERTEKV DEFINED
?
```

A NYOMV segédfunkció a táblázatot soronként kinyomtatja. Bemenetként csak a táblázat nevét kell megadnunk.

A TOLTV segédfunkció lehetővé teszi, hogy rövid felhasználói párbeszéd keretében a táblázat valamennyi mezőjét kitöltsük.

A NYOMV és TOLTV segédfunkciók használják az ismert FOR vezérlő szerkezetet, amellyel könnyen definiálhatjuk az indexes változókat.

Töltsünk most föl egy táblázatot!

```
?DEFV "TAB 2 3

TOLTV "TAB
TAB 1 1 MEZOVEL ...: EN
TAB 1 2 MEZOVEL ...: [1,2 MEZO]
TAB 1 3 MEZOVEL ...: LACI
TAB 2 1 MEZOVEL ...: [A]
TAB 2 2 MEZOVEL ...: .
TAB 2 3 MEZOVEL ...: [A B C]
```

```
?PR OV "TAB 1 2
1,2 MEZO

?NYOMV "TAB
EN 1,2 MEZO LACI
A . A B C

?PONS
VTAB IS [2 3]
VTAB2 IS [[A] . [A B C]]
VTAB3 IS [EN [1,2 MEZO] LACI]
?
```

Mivel a munkatár nem korlátlan, a tömbkezelő funkciókat csomaggá lehet összekötni, és BURY-vel a tárba rejthetjük. Ezeket a funkciókat egyébként lemezen tanácsos tárolni. Helymegtakarítás céljából ezen elemi funkciócsomagból elhagyhatjuk a NYOMV és TOLTV funkciókat.

A tömbkezelő funkciókkal mezőelemként listát is feldolgozhatunk. Így az előzőkben bemutatott táblázatok nem kötődnek egyetlen adattípushoz sem. Ahhoz, hogy a funkciók a listákat is mezőelemként kezelhessék a BEILLESZTL funkcióban gondoskodni kell arról, hogy a lista zárt listaként a táblázatsor eleme legyen. A listák a LISTA?-ban még egy szögletes zárójelbe kerülnek, mivel a SENTENCE egyébként átvenné a listaelemeket, de nem az egész listát. Az elmondottak szemléltetésére próbáljuk ki mindkét változatot, és figyeljük meg, hogy a régi BEILLESZT funkció milyen eredményt ad listák esetén.

```
?
?TO BEILLESZTL :V :S :ERTEK
>IS :S = 1 [OP SE LISTA? :ERTEK BF :V]
>OP SE LISTA? FIRST :V BEILLESZTL BF :V :S - 1 :ERTEK
>END
BEILLESZTL DEFINED
?
?
?TO LISTA? :OBJ
>IF LISTP :OBJ [OP (LIST :OBJ)] [OP :OBJ]
>END
LISTA? DEFINED
?
?
```

4.4.2. Mátrixműveletek

A tömbfunkciók kiegészítéseként még megmutatjuk a mátrixok összeadását, szorzását és transzponálását. Kezdjük két mátrix összeadásával, a művelet eredményét egy harmadik mátrixban tároljuk. Most négyzetes mátrixokkal dolgozunk, azaz minden egyes mátrixban azonos számú sor és oszlop van.

A mátrixösszeadás definíciója a következő:

$$\begin{vmatrix} a_1 & a_2 \\ b_1 & b_2 \end{vmatrix} + \begin{vmatrix} c_1 & c_2 \\ d_1 & d_2 \end{vmatrix} = \begin{vmatrix} a_1 + c_1 & a_2 + c_2 \\ b_1 + d_1 & b_2 + d_2 \end{vmatrix}.$$

Az azonos indexű elemeket összeadjuk, ez adja az eredménytáblázat új elemét.

Lássuk a megfelelő MATADD felhasználói funkciót, amelynek bemenete csupán a három mátrix neve. A :C változó az eredménymátrix neve.

```
?
?
?TO MATADD :A :B :C
>FOR "I 1 FIRST DIMV :A 1 [FOR "J 1 LAST DIMV :A 1
>[IV :C :I :J SUM OV :A :I :J OV :B :I :J]]
>END
MATADD DEFINED
?
?
```

Teszteljük az új funkciót, definiáljunk egy 3X3-as mátrixot, amelyet a már ismert módon feltöltünk az 1...9 számokkal. Példánkban az "A mátrixot önmagával adjuk össze, az eredményt tároljuk az "A mátrixban.

```
?DEFV "A 3 3
?TOLTV "A
A 1 1 MEZOVEL ...:1
A 1 2 MEZOVEL ...:2
A 1 3 MEZOVEL ...:3
A 2 1 MEZOVEL ...:4
A 2 2 MEZOVEL ...:5
A 2 3 MEZOVEL ...:6
A 3 1 MEZOVEL ...:7
A 3 2 MEZOVEL ...:8
A 3 3 MEZOVEL ...:9

?NYOMV "A
1 2 3
4 5 6
7 8 9

?MATADD "A "A "A
?NYOMV "A
2 4 6
8 10 12
14 16 18
?
```

Ha egy mátrixot megszorzunk egy számmal, akkor minden egyes elemet meg kell szorozni ezzel a számmal. A bemeneti változók az :A mátrix, :C az eredménymátrix és a :SZAM konstans.

Példánkban a már ismert "A mátrixot 100-zal szorozzuk, az eredményt rögtön "A-ban tároljuk.

```
?
?
?TO MATSZORZ :A :C :SZAM
>FO "I 1 FIRST DIMV :A 1 [FOR "J 1 LAST DIMV :A 1
>IV :C :I :J ( OV :A :I :J ) * :SZAM]]
>END
MATSZORZ DEFINED
?
?
?
?


?MATSZORZ "A "A 100
?NYOMV "A
200 400 600
800 1000 1200
1400 1600 1800
?
```

A mátrixok szorzása már kissé bonyolultabb. Lássuk először a szorzás szabályát!

$$\begin{vmatrix} a_1 & a_2 \\ b_1 & b_2 \end{vmatrix} \times \begin{vmatrix} c_1 & c_2 \\ d_1 & d_2 \end{vmatrix} = \begin{vmatrix} a_1c_1+a_2d_1 & a_1c_2+a_2d_2 \\ b_1c_1+b_2d_1 & b_1c_2+b_2d_2 \end{vmatrix}$$

Az eredménymátrix új elemét az ún. sor–oszlop szorzással kapjuk, ami az első mátrix sorvektorainak a második mátrix oszlopvektoraival való skaláris szorzatát jelenti. Ezt a CELEM funkcióval állítjuk elő, ami csak szorzásokat hajt végre. A CELEM megírására azért volt szükség, mivel az Apple számítógép hosszabb bemeneti sorokat nem enged meg.

Definiáljuk már ismert módon az A, B és C mátrixokat. Tároljuk a szorzás eredményét a C mátrixban. A TOLTV segédrutin felhasználásával a két mátrixot töltsük fel az 1, 2, 3, 4, ill. 5, 6, 7 és 8 számokkal.

```
?
?TO MATSZOR :A :B :C
>FOR "I 1 FIRST DIMV :A 1 [FOR "K 1 LAST DIMV :B 1
>[MAKE "S 0 FOR "J 1 LAST DIMV :A 1
>[MAKE "S :S + CELEM] IV :C :I :K :S ]]
>END
MATSZOR DEFINED
?
?
?TO CELEM
>OP (OV :A :I :J) * (OV :B :J :K)
>END
```

```
CELEM DEFINED
?
?NYOMV "A
1 2
3 4

?NYOMV "B
5 6
7 8

?MATSZOR "A "B "C
?NYOMV "C
19 22
43 50
?
```

A mátrix transzponálása a sorok és oszlopok felcserélését jelenti. Magyarázat helyett lássunk egy egyszerű példát: transzponáljuk az A mátrixot, az eredmény kerüljön a C mátrixba.

A megfelelő MATTRANS funkció a következő:

```
?TO MATTRANS :A :C
>FOR "I 1 FIRST DIMV :A 1 [FOR "J 1 LAST DIMV :A 1
>[IV :C :J :I OV :A :I :J]]
>END
MATTRANS DEFINED
?
?

?NYOMV "A
A B C
D E F
G H I

?DEFV "C 3 3
?MATTRANS "A "C

?NYOMV "C
A D G
B E H
C F I
?
```

4.4.3. Tömbrendezések Bubblesort

A rendező eljárások a számítástechnika alapfeladatai közé tartoznak. Mivel már tömböket is tudunk kezelni, bemutatunk egy programpéldát anélkül, hogy részletesen foglalkoznánk a rendező eljárásokkal és ezek magyarázatával.

A tömbrendező algoritmusok egydimenziós indexes változókkal dolgoznak. Ez azt jelenti, hogy a táblázat- vagy mátrixfunkcióink különleges esetével van dolgunk, hiszen a táblázat csupán egy sorból áll.

A Bubblesort szemléltetésére lássuk egy tízelemű lista rendezését. Indítsuk

meg ezután a Bubblesort-ot. A lista minden egyes változását kiíratjuk, ily módon követhető a Bubblesort munkamódszere.

```
?NYOMV "PROBA
999 981 780 301 223 799 798 600 555 111

?BUBBLE.SORT "PROBA
999 981 780 301 223 799 798 600 535 111

111 999 981 780 301 223 799 798 600 555

111 223 999 981 780 301 555 799 798 600

111 223 301 999 981 780 555 600 799 798

111 223 301 555 999 981 780 600 798 799

111 223 301 555 600 999 981 780 798 799

111 223 301 555 600 780 999 981 798 799

111 223 301 555 600 780 798 999 981 799

111 223 301 555 600 780 798 799 999 981

111 223 301 555 600 780 798 799 981 999
?
?
```

Meghatározzuk a legkisebb elemet, és ezt legelőre visszük. A lista rendezetlen részének mindenkori legkisebb elemei úgy szállnak fel a felszínre (esetünkben a lista bal szélére), mint a buborékok. A programokat minden további magyarázat nélkül kilistázzuk:

```
?
?
?TO BUBBLE.SORT :NEV
>LOCAL "I MAKE "I 0
>LOCAL "VEGE MAKE "VEGE LAST DIMV :NEV
>REPEAT :VEGE [MAKE "I :I + 1 LEGKISEBB.ELORE :VEGE :I]
>END
BUBBLE.SORT DEFINED
?
?
?TO LEGKISEBB.ELORE :VEGE :KEZD
>LABEL "START
>IF (OV :NEV 1 :VEGE - 1) > (OV :NEV 1 :VEGE)
 [CSERE] MAKE "VEGE :VEGE - 1
>IF :VEGE > :KEZD [GO "START]
>END
LEGKISEBB.ELORE DEFINED
```

```
?
?
?TO CSERE
>LOCAL "H MAKE "H OV :NEV 1 :VEGE
>IV :NEV 1 :VEGE OV :NEV 1 (:VEGE - 1)
>IV :NEV 1 :VEGE - 1 :H
>END
CSERE DEFINED
?
```

A rendszeres nyomtatást azzal értük el, hogy a REPEAT utasítássorba beillesztettük az ismert NYOMV segédrutint.

A BUBBLESORT természetesen szavakat is rendez, ha beépítünk egy megfelelő GREATER vizsgáló funkciót, és megfelelően beépítjük a BUBBLE.SORT-ba.

A GREATER vizsgáló szó a SZ1.MOGE.SZ2? műveletet használja. Az eredményt NOT-tal reagálni kell, ellenkező esetben az nem tűnne nagyobbnak.

```
?
?TO SZ1.MOGE.SZ2 :SZ1 :SZ2
>IF EMPTYP :SZ1 [OP "TRUE]
>IF EMPTYP :SZ2 [OP "FALSE]
>IF FIRST :SZ1 = FIRST :SZ2 [OP SZ1.MOGE.SZ2 BF :SZ1
>BF :SZ2] OP ASCII FIRST :SZ1 < ASCII FIRST :SZ2
>END
SZ1.MOGE.SZ2 DEFINED
?

?NYOMV "PROBA
ANTAL GYULA LAJOS TE EMIL FTC BM EN ALIZ ZOO

?BUBBLE.SORT "PROBA

?NYOMV "PROBA
ALIZ ANTAL BM EMIL EN FTC GYULA LAJOS TE ZOO
?
```

További rendezőalgoritmusok és részletes elemzésük Pascal változatban Wirt, N.: Algoritmusok + Adatstruktúrák = Programok c. könyvében találhatók meg.

Függelék

1. Az LCSI Apple Logo alapfunkciói

2. A Commodore 64 Terrapin Logo leírása

3. A Snail Logo Sinclair ZX Spectrumra

4. Az egyes fejezetek szimbólumai és a hozzá tartozó programok

5. Fordítóprogram

6. Személyi számítógépek Logo változatainak táblázata

1. Az LCSI Apple Logo alapfunkciói

1.1. Aritmetikai műveletek

ARCTAN
Paramétere egy szám, eredménye e szám arcustangense.

COS
Paramétere egy szám, eredménye ezen szám koszinusza.

INT
Paramétere egy szám (decimális szám), eredménye ennek egész része.

PRODUCT*
Paramétere két szám, eredménye ezen számok szorzata.

QUOTIENT
Paramétere két szám, eredménye az első szám második számmal való osztásának egész része.

RANDOM
Paramétere egy nem negatív szám, eredménye egy pozitív, a paraméternél kisebb egész szám.

REMAINDER
Paramétere két szám, eredménye a két szám osztásának maradéka.

RERANDOM
A RANDOM-mal megismételteti ugyanazt a véletlenszám-sorozatot.

ROUND
Paramétere egy szám, eredménye a bemenet legközelebbi egész számra kerekített értéke.

SIN
Paramétere egy szám, eredménye ennek a szinusza.

SQRT
Paramétere egy pozitív szám, eredménye ennek négyzetgyöke.

SUM (SUM...)
Paramétere két szám, eredménye ezen számok összege.

A következő műveleti jeleket infixes alakban használjuk, azaz a műveleti jeleket a paraméterek közé kell írni.

+

Paramétere két szám, eredménye ezen számok összege.

–

Paramétere két szám, eredménye ezen számok különbsége.

/

Paramétere két szám, eredménye a hányados.

<

Paramétere két szám, összehasonlítja, hogy az első szám kisebb-e a másodiknál. Eredménye "TRUE vagy "FALSE.

=

Paramétere két szám, amelyekről megvizsgálja, hogy azonosak-e. Eredménye "TRUE vagy "FALSE.

>

Paramétere két szám, összehasonlítja, hogy az első szám kisebb-e a másodiknál. Eredménye "TRUE vagy "FALSE.

1.2. Képernyőfunkciók

CLEARTEXT
Törli az egész képernyőt.

CURSOR
Kételemű listaként [oszlop–sor] megadja a kurzor pozícióját.

FULLSCREEN
A teljes képernyő a rajzmező rendelkezésére áll.

SETCURSOR
Paramétere egy kételemű lista [oszlop–sor]. A kurzort a megadott helyre állítja.

SPLITSCREEN
A képernyőt egy felső, grafikai részre és egy alsó, négysoros szövegrészre osztja.

TEXTSCREEN
Az egész képernyő a szöveg rendelkezésére áll. Átkapcsolás a grafikus üzemmódból a szöveges üzemmódba.

1.3. Definiálás és editálás

EDIT
(ED)
Paramétere egy szó. A szónak megfelelő nevű funkció editálására hívja a Logo editort.

EDNS
Paramétere egy szó vagy egy lista. A szó vagy a lista szavai azon csomagok nevei, amelyek változóit módosítani kell.

TO
Paramétere egy név, ezt követik a bemeneti paraméterek. Hatására a megadott névvel funkciót lehet definiálni. A definíció END-del fejezhető be.

COPYDEF
Paramétere két szó, ezek funkciónevek. Az első szónak megfelelő névvel másolat készül a második szóval megadott nevű funkcióról.

DEFINE
Paramétere egy szó vagy egy lista. A szó egy funkciónév kell, hogy legyen, amelynek tartalma az őt követő listában levő utasítás.

TEXT
Az adott nevű felhasználói funkció definícióját listaként adja meg.

1.4. Tulajdonságlisták

GPROP
Paramétere egy szó (név) és egy második szó (a tulajdonság megnevezése). A névvel jelölt tulajdonságlistában levő tulajdonság értékét adja meg.

PLIST
Paramétere egy szó (név), eredménye a szóval jelölt tulajdonságlista.

PPROP
Paramétere egy név (szó), egy tulajdonság megjelölés (szó) és a tartalom (objektum), ezeket a tulajdonságokat beírja a tulajdonságlistába.

PPS
Paramétere egy szó vagy egy lista. A szó vagy a szavak csomagok nevei, ezek tulajdonságait kell kinyomtatni.

REMPROP
Paramétere egy név (szó) és egy tulajdonságmegjelölés (szó), a megjelölt tulajdonságlistából törli a megadott tulajdonságot.

1.5. Ki- és bemeneti utasítások

BUTTONP
L. a vizsgáló műveleteket!

KEYP
L. a vizsgáló műveleteket!

PADDLE
Paramétere egy szám, eredménye a számmal jelölt Paddle forgásának mértéke.

.PRINTER
Paramétere egy szám. Adatkivitel a nyomtatóra. A beírt szám a nyomtató csatlakozószáma.

PRINT
(PR)
Paramétere egy objektum, az objektumot – és egy RETURN jelet – nyomtat ki. A listák külső zárójeleit nem írja ki.

READCHAR
(RC)
Addig vár, míg megnyomjuk a billentyűzet egy gombját, eredménye ez a jel.

READLIST
Addig vár, amíg a begépelt sort RETURN-nel lezárjuk. Eredménye a felhasználó által begépelt sor.

SHOW
Ugyanaz, mint a PRINT. Ráadásul kinyomtatja a listák külső szögletes zárójeleit.

TYPE (TYPE...)
Paramétere egy objektum, amelyet kinyomtat. A listák külső zárójeleit nem írja ki. A TYPE után nem következik RETURN, így több TYPE utasításból egyetlen sor keletkezik.

1.6. Listákat és szavakat kezelő funkciók

ASCII
Paramétere egy jel. Eredménye az ASCII kódtáblázat szerinti decimális kód.

BUTFIRST
(BF)
Paraméterét az első elem nélkül adja vissza eredményül.

BUTLAST
(BL)
Paraméterét az utolsó elem nélkül adja vissza eredményként.

CHAR
Paramétere egy szám, eredménye az a jel, amelynek ASCII kódja felel meg az adott számnak.

COUNT
Eredménye a paraméterlistában levő elemek száma.

EMPTYP
L. a vizsgáló műveleteket!

EQUALP
L. a vizsgáló műveleteket!

FIRST
Eredményként a paraméter első elemét adja.

FPUT
Paramétere egy objektum és egy lista, eredménye egy olyan lista, amelyben az objektum a lista elején van.

ITEM
Paramétere egy szám és egy lista vagy szó, eredménye az adott szám szerinti n-edik elem.

LAST
Eredménye a paraméter utolsó eleme.

LIST (LIST...)
Paramétere két objektum, eredménye a paraméterekből készült lista.

LISTP
L. a vizsgáló műveleteket!

LPUT
L. az FPUT-ot! Az elem a lista végére kerül.

MEMBERP
L. a vizsgáló műveleteket!

NUMBERP
L. a vizsgáló műveleteket!

SENTENCE
(SE)
Paramétere két objektum. Eredménye a paraméterekből képzett lista.

WORD
Paramétere két szó. Eredménye a két szóból képzett szó.

WORDP
L. a vizsgáló műveleteket!

1.7. Vezérlő szerkezetek

CATCH
Két paramétere van – egy szó mint név, és egy lista. A listában levő utasítások tartalma végrehajtódik. A név azt a belépési pontot jelöli, amelyre az azonos nevű THROW-val hivatkozni lehet.

CO
A CONTINUE helyett áll, a program futását a PAUSE-zal előidézett megállás után folytatja. CO-t csak a kezelő viheti be a billentyűzettel.

ERROR
A művelet eredménye egy hibaleírásokból álló lista, ha korábban valamilyen programhiba volt. Ellenkező esetben a ERROR eredménye egy üres lista. Hibakimenetekkel kapcsolatban használják.

GO
Paramétere egy szó, amely ugyanezen felhasználói funkcióban levő belépési pontot adja meg. A belépési pontot LABEL-lel és ugyanezen névvel kell jelölni.

IF
Paramétere egy vizsgálandó állítás ("TRUE vagy "FALSE); ezt követi egy vagy két, utasításokból álló lista. A vizsgálati megállapítás logikai műveletek eredménye. TRUE esetben a gép az első utasításlistát hajtja végre, FALSE esetben a másodikat. Az IF egy utasítássorral is előfordulhat, amelynek végrehajtására csak a TRUE esetben kerül sor.

IFFALSE
(IFF)
Paramétere egy utasításokat tartalmazó lista, amelynek végrehajtására akkor kerül sor, ha TEST-tel előzőleg vizsgáló műveletet hajtunk végre és annak eredménye "FALSE.

IFTRUE
(IFT)
Mint az IFFALSE. Az utasításlista végrehajtására csak a TRUE esetben kerül sor.

LABEL
Paramétere egy szó, amely a hozzá tartozó GO belépési pontját adja meg.

OUTPUT
(OP)
Paramétere egy szó vagy egy lista, felhasználó által definiált funkciókban használják. A vezérlés visszakerül a hívó funkcióhoz, az egyik eredményét magával viszi.

PAUSE
Leállítja a felhasználói funkciót. A program futása csak úgy folytatható, ha a billentyűzetről beírjuk a CO-t.

REPEAT
Paramétere egy szám és egy ezt követő, utasításokat tartalmazó lista. Az utasítások annyiszor hajtódnak végre, amennyit a szám előírt.

RUN
Paramétere egy lista. Végrehajtja a listában levő utasításokat és tovább adja az esetleges eredményt.

STOP
Leállítja a felhasználói funkciót és visszaadja a vezérlést a hívó felhasználói funkciónak.

TEST
Paramétere egy vizsgáló művelet, és a rendszer megjegyzi, hogy az eredmény "TRUE vagy "FALSE volt.

THROW
Paramétere egy szó, amely a hozzá tartozó CATCH jelet azonosítja, és visszaléptet.

TOPLEVEL
A THROW speciális paramétere. A program futása megszakad, és a közvetlen utasításos üzemmódba kerül a gép. Ez megfelel a billentyűzeten beírt CTRL-G hatásának.

1.8. Lemezről olvasás és lemezre írás

CATALOG
Kiírja a lemez tartalomjegyzékét valamennyi fájl felsorolásával.

DISK
Információt nyújt a lemezről (egységszám, a vezérlő csatlakozója és a lemez száma).

ERASEFILE
Paramétere egy név. Ez a név a törlendő fájl neve.

LOAD
Paramétere egy név. Ez a név annak a fájlnak a neve, amelyet a munkaterületre be kell olvasni. A LOAD-nak lehet egy második paramétere is. Ez annak a csomagnak a neve, amelybe a teljes beolvasott anyag tartozik.

SAVE
Paramétere egy név. A tár teljes tartalma ilyen néven kerül lemezre. A SAVE második paramétere egy szó vagy szavakból álló lista lehet. Ezek a szavak csomagnevek, amelyek tartalmát a tárból a gép kiválogatja és lemezen tárolja.

SETDISK
Közönséges esetben paramétere egy szám, amellyel a kívánt lemezegységre hivatkozunk. Ennek segítségével meg lehet adni a felhasználói lemezegység csatlakozószámát és a lemez számát.

1.9. Logikai műveletek

AND
Paramétere a két logikai érték, eredménye "TRUE, ha mindkét érték "TRUE.

NOT
Negálja a paraméterként beírt vizsgálószót vagy logikai értéket.

OR
Paramétere két logikai érték, az eredmény "TRUE, ha az egyik paraméter igaz.

1.10. Vizsgáló műveletek

A vizsgáló műveletek eredménye "FALSE vagy "TRUE. IF, TEST vagy logikai műveletek (AND, NOT, OR) paraméterei lehetnek.

BUTTONP
Paramétere egy szám. Eredménye "TRUE, ha a számmal jelölt PADDLE gombját nyomták le.

DEFINEDP
Paramétere egy szó, amely megvizsgálja, hogy ilyen néven definiáltunk-e funkciót.

EMPTYP
Megvizsgálja, hogy paramétere üres szó, vagy üres lista-e.

EQUALP
Megvizsgálja, hogy a két paramétere azonos-e.

KEYP
Ha a billentyűzeten egy gombot megnyomunk és/vagy a karakterpufferben vannak-e jelek, akkor az eredmény "TRUE.

LISTP
Megvizsgálja, hogy paramétere lista-e.

MEMBERP
Paramétere egy elem és egy lista. Megvizsgálja, hogy az elem hozzátartozik-e a listához.

NAMEP
Megvizsgálja, hogy paramétere egy funkció neve-e.

NUMBERP
Megvizsglája, hogy a paraméter szám-e.

PRIMITIVEP
Megvizsgálja, hogy a beírt paraméterszó egy Logo funkció neve-e.

WORDP
Megvizsgálja, hogy a paraméter szó-e.

1.11. Rendszerfunkciók

Részletes ismertetése a gyártói kézikönyvben található.

NODES
Megadja, hogy a munkatárban hány szabad csomópont van.

RECYCLE
Garbage–Collection-futást indít.

REPARSE
Reparsing–futást indít.

.BPT
Belépés az Apple monitorba.

.CONTENTS
Megadja a rendszer által ismert nevek és funkciók listáját.

.DEPOSIT
Az elsőként megadott decimális számon (címen) a második decimális számot tárolja.

.EXAMINE
Paramétere egy decimális cím, eredménye az ott tárolt érték.

1.12. Teknőc-grafika

BACK
(BK)
Paramétere egy szám, amely megadja azoknak a lépéseknek a számát, amelyeket a Teknőcnek hátrafelé kell megtennie.

BACKGROUND
(BG)
Jelzőszám formájában megadja a képernyő háttérszínét.

CLEAN
Törli a rajzmezőt anélkül, hogy megváltoztatná a Teknőc helyzetét.

CLEARSCREEN
(CS)
Törli a képernyőt és a Teknőcöt észak felé nézve (360°) a [0 0] helyzetbe állítja.

DOT
Paramétere egy kételemű lista, [*x* koordináta *y* koordináta], a megadott helyre pontot ír.

FENCE
A Teknőc csak a képernyőn belül mehet.

FORWARD
(FD)
Paramétere egy szám, amely megadja az előre megteendő lépések számát.

HEADING
A Teknőc irányát 0...360° közé eső számmal megadja.

HIDETURTLE
(HD)
A Teknőcöt láthatatlanná teszi.

HOME
A Teknőcöt a [0 0] pozícióba, észak felé nézve állítja be.

LEFT
(LT)
Paramétere egy szám. A Teknőcöt az adott számú fokkal balra fordítja.

PEN
Eredménye egy szám, amely az írótoll állapotát írja le. (PENDOWN, PENERASE, PENUP vagy PENREVERSE, továbbá a szín mutatószáma).

PENCOLOR
Eredménye egy szám, amely az írótoll színét adja meg.

PENDOWN
(PD)
Az írótollat lerakja, így a Teknőc mozgása során nyomot hagy.

PENERASE
(PE)
A Teknőc az útjába kerülő valamennyi vonalat törli.

PENREVERSE
(PX)
Az írótoll fordító hatású, azaz negatív vonalak keletkeznek (fehéren fekete).

PENUP
Az írótollat felemeljük, azaz a Teknőc mozgása során nem hagy nyomot.

POS
Lista alakjában [*x* koordináta, *y* koordináta] megadja a Teknőc helyzetét.

RIGHT RT
Paramétere egy szám. A Teknőcöt az adott fokkal jobbra fordítja.

SCRUNCH
Eredménye egy szám, amely a képernyő leképezési arányát a függőleges lépés–vízszintes lépés arányaként adja meg.

SETBG
Paramétere egy szám. Ettől a számtól függ a háttérszín.

SETHEADING
(SETH)
Paramétere egy szám. A Teknőc a szögtől függően helyezkedik el.

SETPC
Paramétere egy szám. Az írótoll színe ettől a számtól függ.

SETPEN
Paramétere egy lista. Megszabja az írótoll színét és a Teknőc állapotát (l. a PEN-t).

SETPOS
Paramétere egy lista. A Teknőc a megadott koordináták szerinti pozíciót foglalja el.

SETSCRUNCH
Paramétere egy szám. Megszabja a képernyőn a leképezési arányt. Torzításmentes értékek a 0,8...1,2 tartományba esnek a képernyőtől függően.

SETX
Paramétere egy szám. A Teknőc a megadott X koordináta szerinti pozíciót foglalja el.

SETY
Paramétere egy szám. A Teknőc a megadott Y koordináta szerinti pozíciót foglalja el.

SHOWNP
Az eredménye "TRUE, ha a Teknőc látható.

SHOWTURTLE
(ST)
A Teknőcöt ismét láthatóvá teszi. Megszünteti a HIDETURTLE hatását.

TOWARDS
Paramétere a két koordináta, eredménye az az irány fokokban, amelyet a Teknőc helyzete és az adott pont határoz meg.

WINDOW
A rajzmező korlátlanná válik, de látni csak a képernyő ablakába eső pontjait lehet.

WRAP
A rajzmezőt a határokon egymásba olvasztja. A függőleges és vízszintes képernyőhatárok egybe esnek.

XCOR
Megadja a Teknőc pillanatnyi helyzetének *x* koordinátáját.

YCOR
Megadja a Teknőc pillanatnyi helyzetének *y* koordinátáját.

1.13. A munkaterület kezelése

BURY
Paramétere egy szó. Ez a szó annak a csomagnak a neve, amelynek tartalmát (funkciókat és változókat) elrejtjük. A POTS és PONS alkalmazásakor ezek nem listázódnak ki, és nem érinti őket a SAVE utasítás.

ERALL
Paramétere egy szó vagy egy szavakból álló lista. A szavak olyan csomagok nevei, amelyeknek tartalmát törölni kell. (Funkciók és változók.)

ERASE
(ER)
Paramétere egy szó vagy egy szavakból álló lista. A szavak funkciók nevei. A felsorolt funkciók törlődnek.

ERN
Mint az ERASE. Törli a felsorolt neveket, amelyek változók.

ERNS
Mint az ERALL. A felsorolt csomagokban csak a változók törlődnek.

ERPS
Mint az ERNS. A felsorolt csomagokban minden felhasználói funkció törlődik.

PACKAGE
Paramétere egy csomagnév és egy funkciónév. A felsorolt funkció az adott csomagba kerül.

PKGALL
Paramétere egy csomagnév, és a munkatár teljes tartalmát az adott csomagba rakja. Kivételt képeznek a már meglevő csomagok.

PO
Paramétere egy szó vagy egy szavakból álló lista. A szavak funkciók nevei. A megadott funkciók kilistázódnak.

POALL
Paramétere egy szó vagy egy szavakból álló lista. A szavak csomagok nevei. Kilistázódik a csomagok tartalma (funkciók vagy változók). Paraméter hiányában a funkció a teljes munkatárra érvényes.

PONS
Mint a POALL. Csak a változókat listázza ki.

POPS
Mint a POALL. Csak a definiált funkciókat listázza ki.

POTS
Mint a POALL. A funkciók fejlécsorát listázza ki.

UNBURY
Megszünteti a BURY hatását.

1.14. Értékadás (változók definiálása)

LOCAL
Paramétere egy szó. Az így megnevezett változó lokális érték lesz.

MAKE
Paramétere egy szó és egy objektum (szám, szó vagy lista). A szó a változó neve, tartalma az objektum lesz.

NAME
Mint a MAKE. A paraméterek sorrendje fordított.

NAMEP
L. a vizsgáló műveleteket!

THING (:)
Paramétere egy szó. Eredménye a szóval jelölt változó tartalma.

1.15. További fenntartott Logo szavak

END
Jelzi a rendszernek, hogy valamely funkció definiálása befejeződött (TO funkcióval együtt).

ERRACT
Amikor ennek értéke "TRUE, akkor hiba van, és a Logo leáll.

FALSE
Speciális szó, az AND, NOT, OR, IF és TEST műveletek eredménye.

PROCPKG
Egy olyan felhasználói funkció nevének tulajdonsága, amelyhez csomagnév tartozik.

REDEFP
Ha ez a rendszerváltozó "TRUE, akkor a Logo funkciók átdefiniálhatók.

STARTUP
Ha a rendszerváltozó egy lista, akkor a rendszer betöltése után ez a lista hajtódik végre.

TOPLEVEL
Jelölés a THROW számára, a közvetlen utasításos üzemmódba kell áttérni.

TRUE
Speciális szó, az AND, NOT, OR, IF és TEST műveletek eredménye.

VALPKG
Olyan változó nevének tulajdonsága, amelyhez csomagnév tartozik.

.SYSTEM
Az a csomag, amely az ERRACT-ot és a REDEFP-t tartalmazza.

2. A Commodore 64 Terrapin Logo leírása

Jelölések:
- * jellel jelölt utasítások a korábbi betűvel jelzett verziókból hiányoznak.
- ⟨⟩ jelek közötti szöveget nem kell betűnként begépelni, csupán meg kell keresni a billentyűzeten ezzel a szöveggel ellátott gombot és azt kell lenyomni.
- – jellel elválasztott gombokat egyszerre kell lenyomni.

2.1. Bevezetés

A Logo betöltését lemezről a következő utasítások segítségével érhetjük el.

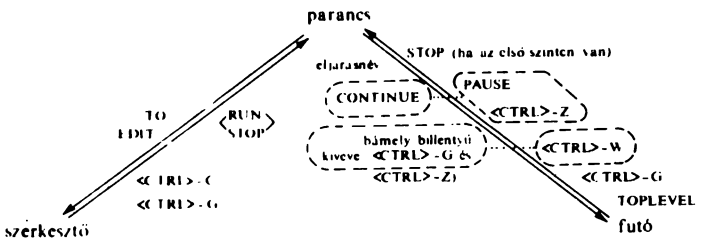
LOAD "LOGO", 8 ⟨RETURN⟩.

A kurzor villogása után a következő parancsot adjuk

RUN ⟨RETURN⟩.

A munkát csak akkor kezdhetjük meg, amikor a Logo bejelentkezett WELCOME TO LOGO! felirattal. A ? mindig azt jelenti, hogy a rendszer új parancsot vár. A villogó négyzet pedig a kurzor, amely a következő kiírásra kerülő szöveg helyét mutatja.

A rendszernek 3 állapota lehetséges:



A rendszer mind parancs-, mind futóállapotban lehet akár grafikus, akár nemgrafikus módban (l. Áttérések).

A Logo felismeri és elfogadja a Commodore billentyűzetének összes betűjelét (a grafikus jeleket is). A felső jelkészlet a ⟨SHIFT⟩ és a megfelelő billentyű egyidejű lenyomásával érhető el, a grafikus jeleket is a ⟨SHIFT⟩ lenyomásával kapjuk.

2.1.1. Funkciók szerkesztőállapotban

⟨CRSR/⇑, ⇓, ⇒, ⇐⟩
A kurzort egy pozícióval fel, le, balra, ill. jobbra mozgatja anélkül, hogy bármit törölne. Folytonos nyomásra ismétlődik.

[↓, ill. ←]
A kurzort egy pozícióval le, ill. balra mozgatja anélkül, hogy bármit törölne. Nem ismétel.

⟨DEL⟩
A kurzortól közvetlenül balra levő jelet törli és egy pozícióval balra lép.

CTRL -A
A kurzort a sor elejére állítja.

CTRL -B
Több képernyőoldalnyi szöveg szerkesztésénél a kurzort egy oldallal hátrébb helyezi vagy a szöveg elejére, ha nincs sok szöveg a kurzor előtt.

CTRL -C
Kilép szerkesztőállapotból és definiálttá teszi az eljárást (eljárásokat). Ugyanezt a hatást lehet elérni a ⟨RUN/STOP⟩ gomb lenyomásával.

⟨CTRL⟩-D
Törli a kurzor pozícióján levő jelet.

⟨CTRL⟩-F
Több képernyőoldalnyi szöveg szerkesztésénél a kurzort egy oldallal előbbre helyezi vagy a szöveg végére, ha nincs sok szöveg a kurzor után.

⟨CTRL⟩-G
Kilép szerkesztőállapotból anélkül, hogy definiálttá tenné az eljárást (eljárásokat). Javítás esetén az eljárás (eljárások) tartalma változatlan marad.

⟨CTRL⟩-K
A kurzortól jobbra eső jeleket törli a sor végéig.

⟨CTRL⟩-L
A kurzort a sor végére mozgatja.

⟨CTRL⟩-N
A kurzort egy sorral lejjebb viszi. Hatása megegyezik a ⟨CRSR/⇓⟩ lenyomásával.

⟨CTRL⟩-O
Üres sort szúr be a kurzor helyén.

⟨CTRL⟩-P
A kurzort egy sorral feljebb viszi. Hatása megegyezik a ⟨CRSR/⇑⟩ lenyomásával.

⟨HOME⟩
A szöveget úgy helyezi el a képernyőn, hogy a kurzornál levő sor a képernyő közepén legyen, kivéve, ha kevés a szöveg.

⟨RETURN⟩
Az aktuális sort a kurzor helyén befejezi, a következő sor elejére téve a kurzort az utána levő szöveggel együtt (ha volt).

⟨RUN/STOP⟩
Kilép a szerkesztőből és definiálttá teszi az eljárást (eljárásokat). Ugyanezt a hatást lehet elérni a ⟨CTRL⟩-C lenyomásával.

⟨SHIFT⟩-⟨INST⟩
CTRL vagy más funkciók (kivéve a ⟨RETURN⟩) előtt alkalmazva, azt engedélyezi beszúrni egy jelsorozatba.

40 jelnél hosszabb sorok végére ! kerül és folytatódnak a következő soron.

2.1.2. Funkciók parancsállapotban

A kurzort egy pozícióval balra, ill. jobbra mozgatja anélkül, hogy bármit törölne. Folytonos nyomásra ismétlődik. A kurzort egy pozícióval balra mozgartja anélkül, hogy bármit törölne.

⟨DEL⟩
A kurzortól közvetlenül balra eső jelet törli és a kurzor egy pozíciójával balra lép.

⟨CTRL⟩-A
A kurzort a sor elejére állítja.

⟨CTRL⟩-D
Törli a kurzor pozícióján levő jelet.

⟨CTRL⟩-K
A kurzortól jobbra eső jeleket törli a sor végéig.

⟨CTRL⟩-P
Az utoljára beírt sort írja ki a kurzortól kezdve. Ugyanezt a hatást lehet elérni a ⟨CRSR/⇑⟩ gomb lenyomásával.

⟨RETURN⟩
Végrehajtja a kurzor környezetében levő utasításokat ?-től az utasítások végéig. (Maximum 255 karakter hosszú utasítássort.)

⟨SHIFT⟩-⟨CLR⟩
Kilép az éppen beírás alatt levő sorból, törli a szövegmező tartalmát és a kurzort a bal felső sarokba teszi.

⟨SHIFT⟩-⟨INST⟩
CTRL vagy más funkciók (kivéve a ⟨RETURN⟩) előtt alkalmazva, azt engedélyezi beszúrni egy jelsorozatba.

2.1.3. Funkciók futóállapotban

⟨CTRL⟩-G
Leállítja az éppen aktív eljárás végrehajtását és a rendszer parancsállapotba tér át.

⟨CTRL⟩-W
Felfüggeszti az éppen aktív eljárás végrehajtását, amelyet bármely gomb (kivéve a ⟨CTRL⟩-G vagy ⟨CTRL⟩-Z) lenyomására tovább folytat.

⟨CTRL⟩-Z
Felfüggeszti az éppen aktív eljárás végrehajtását, amelyet a CONTINUE utasítás hatására folytat.

Az utasítások jelölése.

PENCOLOR *s*
(PC)
S∈[−1, 15] egész
FIRST $sjl_1 \rightarrow sjl_2$
sjl_2:
ANYOF $t_1\ t_2\ \ldots \rightarrow t_3$,

ahol
- PENCOLOR paramétere *s*, amely −1 és 15 közötti egész szám lehet, az utasítás rövidíthető PC *s*-sel.
- FIRST paramétere szám, jelsorozat vagy lista; eredménye is szám, jelsorozat vagy lista, amelynek tartalma a magyarázatban sjl_2 : után szerepel.
- ANYOF paramétere két logikai érték, de több paramétere is lehet, ha kerek zárójelek közé tesszük az egész utasítást. Ilyenkor a zárójel és a paramétere között célszerű egy helyközt kihagyni elkülönítés céljából.

Rövidítések:
- *s* szám (l. Műveleti jelek);
- *j* jelsorozat, " után tetszőleges betűk, számok vagy egyéb jelek;
- *l* lista, [ ] zárójelek között tetszőleges jelsorozatok (" nélkül) helyközzel elválasztva;
- *t* logikai érték, "TRUE vagy "FALSE.

A számok és az őket ábrázoló jelsorozatok ekvivalensek lehetnek:

pl.: 3615 ekvivalens a "3615 jelsorozattal;
4.25 ekvivalens a "4.25 jelsorozattal.

A számok azonban csak belső ábrázolásuk jelsorozatával ekvivalensek mindig:

pl.: 0.005E24 belső ábrázolása valójában 4.999999E21 így jelsorozatuk nem azonos,

ezért óvatosan kell bánni a jelsorozatként alkalmazott számokkal, sőt lehetőleg kerüljük is el használatukat. Ezek a paraméterek nemcsak az előbb leírt alakúak lehetnek, hanem a megfelelő típusú eredményt adó alap- vagy létrehozott eljárások, ill. értelemszerű kifejezések is szerepelnek helyettük.

A paramétereket helyközzel választjuk el egymástól. Egy utasítássor akár utasítássorozatot (utasítások helyközzel elválasztva) is tartalmazhat.

Az utasítássor befejezését a ⟨RETURN⟩ gomb lenyomásával jelezzük.

A megjegyzéseket az utasítások után ; jellel elválasztva írjuk, amelyek nem kerülnek végrehajtásra.

Például:

```
PRINT FIRST 8342 PRINT "NYOLC ; PRINT "POLC
```

eredménye: 8
NYOLC.

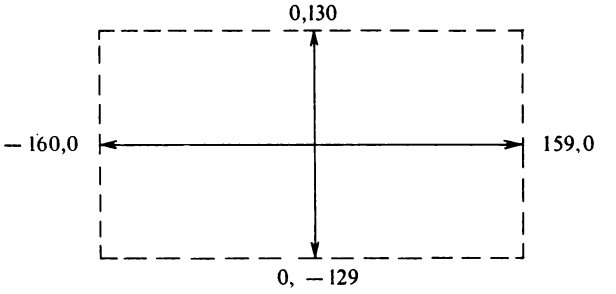
A végrehajtás balról jobbra történik. Az egyes utasítások csak akkor hajtódnak végre, ha a paramétereik már kiértékelődtek. Így az első PRINT utasítás csak a FIRST végrehajtása után tudja meg, hogy a paramétere 8 (8342 első jele) Ezután a második PRINT utasítás azonnal leírja paraméterét, amely a NYOLC jelsorozat. A harmadik PRINT utasítás azonban már nem kerül végrehajtásra, mert a ; jel után mindent megjegyzésnek számít a sor végéig. A : jel értelme attól függ, hogy egy " jel nélküli jelsorozat elején vagy végén áll:

- elején: a jelsorozat változó, amelynek tartalmát adja értékül;
- végén: a jelsorozatot címkének tekinti, ha az egy utasítás elején áll.

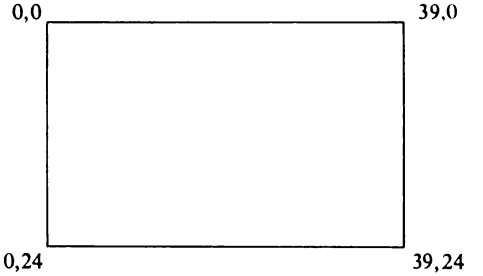
2.2. Képernyőkezelés

2.2.1. Információk

Rajzmező-koordináták:



Szövegmező-koordináták:



Színkódok:

0 fekete;
1 fehér;
2 piros;
3 enciánkék;
4 lila;
5 zöld;
6 kék;
7 sárga;
8 narancssárga;
9 barna;
10 világospiros;
11 sötétszürke;
12 középszürke;
13 világoszöld;
14 világoskék;
15 világosszürke.

2.2.2. Rajzmező – Teknőc-utasítások

BACK *s*
(BK)
A Teknőc irányával ellentétesen halad a képernyőn adott *s* lépést, az aktuális tollszínnek megfelelő csíkot húzva maga után (vagy radíroz), ha a toll lent van. Negatív *s* ellentétes mozgást vált ki.

FORWARD *s*
(FD)
A Teknőc irányával megegyzően halad a képernyőn adott *s* lépést, az aktuális tollszínnek megfelelő csíkot húzva maga után (vagy radíroz), ha a toll lent van. Negatív *s* ellentétes mozgást vált ki.

HEADING → *s*
s: [0, 360] közé eső egész szám, a Teknőc pillanatnyi iránya fokban mérve. 0 esetén a Teknőc felfelé néz. Az óramutató járásával megegyezően nő az irány értéke.

HIDETURTLE
(HT)
A Teknőc eltűnik a rajzmezőről, de állapota nem változik meg, sőt a továbbiakban a rajzolás gyorsabb lesz. Mindaddig érvényben marad, amíg SHOWTURTLE utasítást nem adunk (l. DRAWSTATE).

HOME
A Teknőc a rajzmező közepére megy és felfelé néz. Ha a toll lent van, úgy útközben az aktuális tollszínnek megfelelő csíkot húz maga után (vagy radíroz).

LEFT *s*
(LT)
A Teknőc irányát *s* fokkal balra fordítja. Negatív *s* ellentétes mozgást vált ki.

PENCOLOR *s*
(PC)
$s \in [-1, 15]$ egész szám
A Teknőc rajztollának színét *s* kódura állítja. –1 értékű *s* esetén a toll radírként viselkedik és az utasítás hatása egyenértékű lesz a PENERASE-zel. Alapállapot: 1 (fehér). Mindaddig ebben az állapotban marad, amíg más értelmű PENCOLOR (ill. PENERASE) utasítást nem adunk (l. színkódok, DRAWSTATE).

PENDOWN

(PD)

A Teknőc tollát leereszti, haladáskor a toll az aktuális tollszínnel csíkot húz (ill. radíroz). Alapállapot. Mindaddig ebben az állapotban marad, amíg PENUP utasítást nem adunk (l. DRAWSTATE).

PENERASE

A Teknőc rajztolla radírként viselkedik.

PENUP

(PU)

A Teknőc tollát felemeli, haladáskor a toll nem hagy nyomot. Mindaddig ebben az állapotban marad, amíg PENDOWN utasítást nem adunk (l. DRAWSTATE).

RIGHT *s*

(RT)

A Teknőc irányát *s* fokkal jobbra fordítja. Negatív *s* ellentétes mozgást vált ki.

SETHEADING *s*

(SETH)

A Teknőc irányát *s* fokra állítja. 0 esetén a Teknőc felfelé néz és iránya az óramutató járásával megegyezően nő.

SETX *s*

A Teknőc vízszintesen az adott *s* vízszintes koordinátára megy, az aktuális tollszínnek megfelelő csíkot húzva maga után (ill. radíroz), ha a toll lent van. Az origó a rajzmező közepe.

SETXY s_1 s_2

A Teknőc adott s_1, s_2 koordinátára megy, az aktuális tollszínnek megfelelő csíkot húzva maga után (ill. radíroz), ha a toll lent van. Az origó a rajzmező közepe.

SETY *s*

A Teknőc függőlegesen az adott *s* függőleges koordinátára megy, az aktuális tollszínnek megfelelő csíkot húzva maga után (ill. radíroz), ha a toll lent van. Az origó a rajzmező közepe.

SHOWTURTLE

(ST)

A Teknőc láthatóvá válik a rajzmezőn, de állapota nem változik meg. Alapállapot. Mindaddig ebben az állapotban marad, amíg HIDETURTLE utasítást nem adunk (l. DRAWSTATE).

STAMPCHAR *sj*
s itt csak egyjegyű szám lehet.
j itt csak egyetlen jel lehet.
A Teknőc pillanatnyi pozícióján az aktuális tollszínnel kiírja a rajzmezőre a jel alakját. Ha SINGLECOLOR érvényes, akkor jól láthatóak a jelek.

TOWARDS $s_1\ s_2 \rightarrow s_3$
s_3: [0, 360] közé eső egész szám, ez, az az irány fokban mérve, amellyel a Teknőcnek rendelkeznie kell, hogy egy adott s_1, s_2 koordinátájú pont irányába nézzen. Az origó a rajzmező közepe. 0 fokú irány esetén a Teknőc felfelé néz és iránya az óramutató járásával megegyezően nő.

XCOR $\rightarrow s$
s: a Teknőc pillanatnyi x koordinátája; az origó a rajzmező közepe.

YCOR $\rightarrow s$
s: a Teknőc pillanatnyi y koordinátája; az origó a rajzmező közepe.

2.2.3. Rajzmezőt kezelő utasítások

BACKGROUND *s*
(BG)
$s \in [0, 15]$ egész szám.
A rajzmező háttérszínét az *s* színkódnak megfelelően állítja be. Alapállapot: 11 (szürke). Mindaddig ebben az állapotban marad, amíg más értelmű BACKGROUND utasítást nem adunk (l. színkód, DRAWSTATE).

CLEARSCREEN
(CS)
A rajzmezőt letörli, de a Teknőc állapotát változatlanul hagyja.

DOUBLECOLOR
A rajzmező egy-egy 8×8-as képponttömbjébe ezekután két különböző szín fordulhat elő a háttérszínen kívül. Letörli a rajzmezőt és a grafikus paramétereket alapállapotba állítja. A vízszintes vonalak ezután vastagabbak és nem annyira precízek lesznek. A rajzmezőn kinyomtatott betűk kevésbé felismerhetőek. Mindaddig ebben az állapotban marad, amíg SINGLECOLOR utasítást nem adunk (l. DRAWSTATE).

DRAW
A rajzmezőt letörli és a Teknőcöt a közepére teszi (látható, felfelé néz, rajztolla leeresztve) állapotban. Háttér- és tollszín nem változik.

NOWRAP
A továbbiakban, ha egy Teknőc-művelet a rajzmezőn kívül fejeződne be, úgy hibajelzést ad, és nem hajtja végre a műveletet. Mindaddig ebben az állapotban marad, amíg WRAP, DOUBLECOLOR vagy SINGLECOLOR utasítást nem adunk.

SINGLECOLOR
A rajzmező egy-egy 8×8-as képponttömbjébe ezután csak egy szín fordulhat elő a háttérszínen kívül. Mindig a legutolsó tollszín határozza meg az érintett tömb felgyújtott pontjainak a színét. Letörli a rajzmezőt és a grafikus paramétereket alapállapotba állítja. Vékony és precíz vonalakat eredményez. Alapállapot. Mindaddig ebben az állapotban marad, amíg DOUBLECOLOR utasítást nem adunk (l. DRAWSTATE).

WRAP
A rajzmező körkörösen folytatólagossá válik mind a négy irányban. Ha ezekután egy Teknőc-művelet a rajzmezőn kívül fejeződne be, úgy a szemben levő oldalon folytatja az útját. Alapállapot. Mindaddig ebben az állapotban marad, amíg NOWRAP utasítást nem adunk.

.ASPECT *s*
s-nek megfelelő függőleges rajzmezőarányt állít be. Alapállapot: 0.768. Mindaddig érvényben marad, amíg más értelmű. ASPECT utasítást nem adunk. Megváltoztatásával módosulnak a rajzmező *y* koordinátái. A tv-nek megfelelően kell beállítani, a torzítás elkerülése végett.

2.2.4. Szövegmezőt kezelő utasítások

CLEARTEXT
A szövegmezőt letörli és a kurzort a bal felső sarokba (0, 0) állítja.

CURSOR s_1 s_2
$s_1 \in [0, 39]$ egész szám;
$s_2 \in [0, 24]$ egész szám.
A kurzort az adott s_1 (oszlop), s_2 (sor) pozícióra állítja a szövegmezőn. 0, 0 a bal felső sarok.

CURSORPOS → *l*
l: kételemű lista, amelynek első eleme a kurzor oszlopának, második eleme a sorának pozíciója a szövegmezőn. 0, 0 a bal felső sarok.

NODRAW
(ND)
Áttér nemgrafikus üzemmódba és a grafikus paramétereket alapállapotba

állítja. Az egész (üres) szövegmező láthatóvá válik és a kurzor a bal felső sarokba kerül (l. Áttérések, DRAWSTATE).

TEXTBG *s*
$s \in [0, 15]$ egész szám.
A szövegmező háttérszínét az *s* színkódnak megfelelően állítja be. Alapállapot: 14 (világoskék). Mindaddig ebben az állapotban marad, amíg más értelmű TEXTBG utasítást nem adunk (l. színkód, DRAWSTATE).

2.2.5. Írás a szövegmezőre

FPRINT *sjlt* ....
Kiírja az adott *sjlt* objektumot a kurzor pillanatnyi pozíciójától (több paraméter esetén helyközt hagyva) úgy, hogy a listákat nem fosztja meg a külső [ ] jelektől. A kiírás után a kurzor a következő sor elejére tér.

PRINT *sjlt*...
(PR)
Kiírja az adott *sjlt* objektumot a kurzor pillanatnyi pozíviójától (több paraméter esetén helyközt hagyva) úgy, hogy a jelsorozatok elől elhagyja a " jelet és a listákat megfosztja a külső [ ] jelektől. A kiírás után a kurzor a következő sor elejére tér.

PRINT1 *sjlt*...
Kiírja az adott *sjlt* objektumot a kurzor pillanatnyi pozíciójától (több paraméter esetén szorosan egymás mellé) úgy, hogy a jelsorozatok elől elhagyja a " jelet és a listákat megfosztja a külső [ ] jelektől. A kiírás után a kurzor a kiírt szöveg után marad.

.CTYO *s*
$s \in [0, 255]$ egész szám.
Kiírja az adott *s* ASCII kódú jelet a kurzor pillanatnyi pozíciójára. A kiírás után a kurzor a kiírt jel után marad.

Áttérések:

A rendszer inicializálásakor nemgrafikus módban van és a szövegmező látható kizárólag a képernyőn. Bármely Teknőc-, grafikus, ill. FULLSCREEN, SPLITSCREEN, TEXTSCREEN, DRAWSTATE, READPICT, SAVEPICT utasítás hatására a rendszert áttér grafikus módba és osztott mezőre vált, amelynél a képernyő felső 3/4 része a rajzmezőt mutatja és az első 1/4 része

a szövegmezőt. Osztott mező esetén a kurzor automatikusan a látható szövegmező elejére tér.
Grafikus módban a képernyő átválthat a mezők között.

Grafikus módban

FULLSCREEN röv. (f5)
A képernyő a teljes rajzmezőt mutatja.

SPLITSCREEN röv. (f3)
Osztott mezőre váltott a képernyőnél: 3/4 rész rajzmező, 5 sor szövegmező. Grafikus módban ez az alapállapot.

SPLITSCREEN *s*
$s \in [0, 24]$ egész szám.
Hatására az osztott mezős képernyő *s* sort fog mutatni a szövegmezőből. Mindaddig érvényben marad, amíg más értelmű SPLITSCREEN utasítást nem adunk.

TEXTSCREEN röv. (f1)
A képernyő a teljes szövegmezőt mutatja. A NODRAW utasítás is átvált a teljes szövegmezőre, de az egyben nemgrafikus módba is átviszi a rendszert, amelynél f1, f3, f5 hatástalan marad.

Állapot:

DRAWSTATE → *l*
l: 9 elemű lista, a képernyőre vonatkozó információkat tartalmazza:
1. TRUE (a Teknőc tolla lent van);
   FALSE (a Teknőc tolla fent van):
2. TRUE (a Teknőc látható);
   FALSE (a Teknőc nem látható).
3. [0, 15] közötti egész szám, a rajzmező háttérszínkódja.
4. [−1, 15] közötti egész szám, a Teknőc tollának színkódja.
5. DRAW;
   NODRAW.
6. DOUBLECOLOR;
   SINGLECOLOR.
7. FULLSCREEN;
   SPLITSCREEN;
   TEXTSCREEN.
8. [0, 15] közötti egész szám, a szövegmező háttérszínkódja.
9. [0, 15] közötti egész szám, a szöveg színkódja.

2.3. Műveletvégzés

2.3.1. Műveleti jelek

$s_1 \rightarrow s_3$ aritmetikai műveleti jel; s_2, s_3 aritmetikai műveleti jel: +, –, *, /. Aritmetikai műveleti jelek alkalmazásával kifejezések és részeik zárójelezhetőek kerek zárójelek között. A kifejezésben szereplő negatív számokat zárójelek közé kell tenni.

$s_1 > s_2 \rightarrow t$
t: TRUE, ha s_1 nagyobb, mint s_2
FALSE, különben

$s_1 < s_2 \rightarrow t$.
t: TRUE, ha s_1 kisebb, mint s_2
FALSE, különben

$s_1 = s_2 \rightarrow t$.
t: TRUE, ha s_1 és s_2 numerikus értékei megegyeznek, különben FALSE.

Ha *s* előtt " áll, úgy a jelsorozatot tekinti és nem az értéket. (l. szövegkezelés).

Pl. 2 = "2 eredménye TRUE;
2 = 2.0 eredménye TRUE;
de 2 = "2.0 eredménye FALSE.

Az aritmetikai kifejezések a következő elsőbbségi szabályok szerint értékelődnek ki:

1. zárójelezett kifejezések kiértékelése.
2. * és / kiértékelése balról jobbra.
3. + és – kiértékelése balról jobbra.

Az egyértelműség kedvéért ajánlott a zárójelek használata.

A számok terjedelme függ a típusuktól:

– Valós számok: [1.999 10^{-38}, 1.7*10^{38}]
ábrázolása: [1.999N38, 1.7E38],

ahol N a negatív 10-es alapú hatványkitevőt jelöli, E pedig a pozitívat.

– Egész számoknál: [-214783647, 214783647]

intervallumba eső számokkal tud műveletet végezni, ill. eredményül szolgálni.

Ezen intervallumokon kívül eső számsorozatokat eljárásként értelmezi.

A negatív paramétereket nagy gonddal kell kezelni. A probléma a második vagy még ennél több számú paraméternél adódik, amelyeket zárójelek közé kell tenni.

Például

SETXY 50 –10

végrehajtásánál először az 50–10 művelet értékelődik, ki, amelynek eredménye 40. Ezekután a SETXY 40 utasítás már hiányos, mert az utasítás két paramétert vár. A helyes utasításforma:

SETXY 50 (–10).

2.3.2. Aritmetikai függvények

ATAN s_1 s_2 → s_3
s_3: s_1 és s_2 hányadosának arcusz tangense fokban mérve, amely a két szám előjelétől függő negyedbe esik.

COS s_1 → s_2
s_2: s_1 fok koszinusza.

INTEGER s_1 → s_2
s_2: s_1 egész része.

NUMBER? *sjlt* → *t*
t: TRUE, ha a paraméter *s* (" jel is lehet előtte)
különben FALSE.

QUOTIENT s_1 s_2 → s_3
s_3: s_1 és s_2 kerekített értékeik hányadosának egész része.

REMAINDER s_1 s_2 → s_3
s_3: s_1 és s_2 kerekített értékeik osztásából származó egész maradék.

ROUND s_1 → s_2
s_2: s_1 kerekített értéke.

SIN s_1 → s_2
s_2: s_1 fok szinusza.

SQRT s_1 → s_2
s_2: a pozitív, s_1 pozitív négyzetgyöke.

2.3.3. Véletlenszámok

RANDOM s_1 →s_2
s_1 egész szám
s_2: [0, s_1 – 1] közötti véletlen egész szám.

RANDOMIZE
A véletlenszámgenerátor magját véletlenszerűvé teszi.

RANDOMIZE *s*
A véletlenszámgenerátor magja *s* lesz. Minden végrehajtás után ugyanaz a véletlen számsorozat keletkezik a RANDOM-ok hatására.

2.3.4. Bites műveletek

BITAND s_1 s_2 → s_3
s_1 és s_2 egész számok.
s_3: s_1 és s_2 bináris megfelelőinek bitenkénti "és" művelet 10-es számrendszerbeli megfelelője. Az "és" művelet eredménye egy bitre csak akkor 1, ha mindkét paraméterbit 1, különben Ø.

BITOR s_1 s_2 → s_3
s_1 és s_2 egész számok.
s_3: s_1 és s_2 bináris megfelelőinek bitnkénti "or" művelet 10-es számrendszerbeli megfelelője. Az "or" művelet eredménye egy bitre akkor 1, ha valamelyik paraméterbit 1, különben Ø.

BITXOR s_1 s_2 → s_3
s_1 és s_2 egész számok.
s_3: s_1 és a_2 bináris megfelelőinek bitenkénti " kizáró vagy " művelet 10-es számrendszerbeli megfelelője. A " kizáró vagy " művelet eredménye egy bitre csak akkor 1, ha csak az egyik paraméterbit 1, különben Ø (ha mindkettő Ø, ill. mindkettő 1).

2.4. Szövegkezelés

Megjegyzés:
- A jelsorozat elemei az egyes jelek.
  Például "CICA elemei: C, I, C, A.
- A listák elemei a jelsorozatok, ill. allisták.
  Például [ALMA 3.15 [A B C] –] elemei: ALMA, 3.15, [A B C], –.
- A számok is bizonyos fajta jelsorozatok.
  Például 4.75 elemei: 4 ., 7, 5.
- A logikai értékek elemei az értéküket képviselő jelsorozat elemei.
  Például: "TRUE elemei: T, R, U, E.

Szövegkezelő műveleti jel:

$jlt_1 = jlt_2 \rightarrow t$
t: TRUE, ha jlt_1 és jlt_2 elemei teljesen azonosak
különben FALSE.
(l. műveleti jeleknél '=' értelmét)

2.4.1. Szövegkezelő utasítások

A szövegkezelő utasításoknál is a számok belső ábrázolásuk jelsorozataival azonosak.

Például:
FIRST 34 eredménye: 3.
ITEM 2 4.2 eredménye: .
COUNT 34.56N4 eredménye: 8, mivel 34.56N4 belső ábrázolása Ø.ØØ3456.

Az egyértelműség kedvéért ezért célszerű " jellel reprezentálni a számértelmű jelsorozatokat, ha szövegkezelő utasítások paraméterei.

BUTFIRST *sjlt*$_1$ → *sjlt*$_2$
(BF)
ha *sjlt*$_1$ elemszáma $\geqq 1$.
akkor *sjlt*$_2$: *sjlt*$_1$ első elemén kívüli része

BUTLAST *sjlt*$_1$ → *sjlt*$_2$
(BL)
sjlt$_1$ elemszáma $\geqq 1$
sjlt$_2$: *sjlt*$_1$ utolsó elemén kívüli része

COUNT *sjlt* → *s*
s: *sjlt* elemeinek száma

EMPTY? *sjlt* → *t*
t: TRUE, ha *sjlt*-nek nincs egy eleme sem
pl. üres jelsorozat: " ;
üres lista: [ ].
különben FALSE.

FIRST *sjlt*$_1$ → *sjlt*$_2$;
sjlt$_1$ elemszáma $\geqq 1$;
sjlt$_2$: *sjlt*$_1$ első eleme.

FPUT *sjlt* l_1 → l_2 ;
l_2 : lista, amelynek az első eleme *sjlt*, a többi pedig l_1 elemei.

ITEM *s sjlt*$_1$ → *sjlt*$_2$;
$s \in [1,$ *sjlt*$_1$ elemszáma$]$;
sjlt$_2$: *sjlt*$_1$ *s*-edik eleme.

LAST *sjlt*$_1$ → *sjlt*$_2$;
sjlt$_1$ elemszáma $\geqq 1$;
sjlt$_2$: *sjlt*$_1$ utolsó eleme.

LIST *sjlt*$_1$ *sjlt*$_2$ ... → *l*
l: lista, amelynek elemei sorra *sjlt*$_1$ *sjlt*$_2$ ...

LIST? *sjlt* → *t.*
t: TRUE, ha *l* a paraméter,
különben FALSE.

LPUT *sjlt* l_1 → l_2
l_2 : lista, amelynek az utolsó eleme *sjlt*, a többi pedig l_1 elemei.

MEMBER? *sjlt*$_1$ *sjlt*$_2$ → *t.*
t: TRUE, ha *sjlt*$_1$ eleme *sjlt*$_2$-nek
különben FALSE.

SENTENCE *sjlt*$_1$ *sjlt*$_2$... → *l*
(SE)
l: lista, amelynek elemei sorrendben *sjt*$_1$, ill. l_1 elemei, *sjt*$_2$, ill. l_2 elemei...

WORD *sjt*$_1$ *sjt*$_2$ ... → *sjt*$_3$
sj$_3$: jelsorozat, amelynek elemei sorra *sjt*$_1$ *sjt*$_2$ ...

WORD? *sjlt* → *t*
t: TRUE, ha *jt* a paraméter
különben FALSE.

2.4.2. ASCII konverziók

ASCII *sj* → *s*
s: [0, 255] közötti egész szám, az *sj* jel ASCII kódja.

CHAR *s* → *sj*
$s \in [0, 255]$ egész szám,
j: az *s* ASCII kódhoz tartozó jel, ha van.

2.5. Vezérlésirányítás

2.5.1. Feltételes utasítások

IF *t* THEN utasítások ELSE utasítások
IF *t* utasítások
IF *t* THEN utasítások

– ha *t* értéke TRUE, akkor a THEN utáni (ill. az utána következő, ha nincs THEN) utasítások hatjódnak végre.

– ha *t* értéke FALSE, akkor az ELSE utáni utasítások (ill., ha nincs ELSE ág, úgy a sor végéig semmi sem) hajtódik végre.

TEST *t*
A *t* értékétől függ az utána következő IFFALSE és IFTRUE utasítások kiértékelése.

IFFALSE utasítások
(IFF)
Az utasítások csak akkor hajtódnak végre, ha az utoljára kiértékelt TEST paraméterértéke FALSE volt.

IFTRUE utasítások
(IFT)
Az utasítások csak akkor hajtódnak végre, ha az utoljára kiértékelt TEST paraméterértéke TRUE volt.
Az IFFALSE és IFTRUE tetszőleges mennyiségben és sorrendben fordulhatnak elő a TEST utasítás után ugyanabban az eljárásban.

Feltételek:

Az IF is TEST utasítások paramétertípusa *t*, azaz logikai TRUE (igaz), ill. FALSE (hamis) vagy ilyen értéket adó logikai kifejezés, ill. eljárás eredménye. Logikai értéket adnak a =, < , > műveletek eredményei is.

2.5.2. Logikai utasítások

ALLOF t_1 t_2 ... → t_3
t_3:TRUE, ha a t_1 t_2 ... értékei sorra TRUE-k,
FALSE különben.

ANYOF t_1 t_2 ... → t_3
t_3:TRUE, ha t_1 t_2... értékei közül valamelyik TRUE
FALSE különben.

NOT t_1 → t_2
t_2:TRUE, ha t_1 értéke FALSE,
FALSE, ha t_1 értéke TRUE.

Logikai értéket adnak a ? jellel végződő utasítások is:

EMPTY?, LIST?, MEMBER?, NUMBER?, RC?, THING?, WORD?

Szintén logikai értéket èredményeznek a következő utasítások: JOYBUTTON és PADDLEBUTTON.

2.5.3. Feltétel nélküli vezérlési utasítások

GO *sj*
s pozitív szám.
A vezérlés átadódik eljáráson belül az adott *sj* címkéjű utasításra. Az utasítás címkéket a sor elejére kell tenni: végződéssel.

A valós számokat lehetőleg ne használjuk címkeként, mert a definíciójánál (sor elején :-tal lezárva) úgy tekinti, mint egy fix jelsorozat, de a GO után kiértékeli és a belső ábrázolásának megfelelő jelsorozattal rendelkező címkének értelmezi.

GOODBYE
A Logo inicializálódik és a munkaterület törlődik. Minden eddigi munka elvész (kivéve a felhasználói gépikódú memória területen levők).

OUTPUT *sjlt*
(OP)

Az eljárásból kilépve a vezérlés az előző szintre tér az eljáráshívás helyére és *sjlt*-t az eljárás eredményéül adja. Ha nincs előző szint, úgy egyszerűen kiírja *sjlt*-t mint eredményt. Az ilyen eredményt adó eljárások más eljárások paramétereiként szerepelhetnek.

Például:

```
TO ADD :X :Y
   OUTPUT :X + :Y
END
```

eljárás létrehozása után

```
PRINT 3 * ADD 2 4
```

utasítás eredménye:

```
18.
```

REPEAT *sl*
s pozitív egész szám
l lista, amelynek tartalma maximum 255 jelből álló utasítássorozat.
Az *l*-ben levő utasítássorozatot *s*-szer végrehajtja.

RUN *l*
l egy lista, amelynek tartalma maximum 255 jelből álló utasítássorozat.
Az *l*-ben levő utasítássorozatot egyszer végrehajtja.

STOP
Az eljárásból kilépve a vezérlés visszatér az előző szinten levő eljáráshívás helyére. Ha nincs előző szint, úgy parancs állapotba megy át.

TOPLEVEL
Az eljárásból kilépve parancs állapotba tér át.

2.5.4. A hibakeresés eszközei

CONTINUE
(CO)
A felfüggesztett eljárás végrehajtását folytatja.
Felfüggesztés PAUSE is CTRL-Z hatására történik.

PAUSE
Az eljárás végrehajtását felfüggeszti. Hatása megegyezik a CTRL-Z funkcióval futás közben. A végrehajtás CONTINUE hatására folytatódik.

TRACE
Bekapcsolja a nyomkövető rendszert. Hatására minden utasítás végrehajtása előtt kiírja annak nevét (az eljárásokba való belépéskor a bemenő paraméterek értékeivel együtt) majd leáll és bármely gomb lenyomása esetén (kivéve a CTRL-G és CTRL-Z) a végrehajtás folytatódik.

NOTRACE
Kikapcsolja a nyomkereső rendszert.

Fontos lehet még a CTRL-G és a CTRL-W funkciók hatása futás alatt. Az eljárások felfüggesztésekor ki lehet íratni vagy módosítani változók vagy eljárások tartalmát.

2.6. Bemenő és kimenő jelek

2.6.1. Bemenő jeleket kezelő utasítások

CLEARINPUT
Üresre állítja a karakterpuffert. A lenyomott billentyűk jelei ebbe a pufferbe gyülekeznek (max. 255 jel), amíg a rendszer nem reagál rájuk.

RC? → *t*
t: TRUE, ha egy gomb le lett nyomva, de még nem lett lekezelve különben FALSE.

READCHARACTER → *j*
(RC)
A rendszer gombnyomásra vár.
j: az első lenyomott gomb jele.

REQUEST → *l*
(RQ)
l: lista, amely a ⟨RETURN⟩ gomb lenyomásáig beírt jeleket tartalmazza.

2.6.2. Bemeneti egységek jeleit kezelő utasítások

A botkormány jeleit lekezelő utasítások:

JOYBUTTON *s* → *t*
s [0, 1] egész szám
t: TRUE, ha az *s* kijelölésű botkormány gombja le lett nyomva; különben FALSE.

JOYSTICK $s_1 \rightarrow s_2$
$s_1 \in [0, 1]$ egész
s_2: $[-1, 7]$ közé eső egész szám, az s_1 kijelölésű botkormány iránya:
−1 = közép
0 = Észak
1...7 = É–K, K, D–K, D, D–Ny, Ny, É–Ny.

```
TO INF
 ND
 PR [A BOTKORMANYT "PORT 2"-RE KELL DUGNI.]
 PR [A TOLLAT A BOTKORMANY IRANYITJA ES A]
 PR [BOTGOMB TESZI LE.]
 PR [MUNKA KOZBEN LOGO UTASITASOKAT ELFOGAD.]
 PR [KILEPES A CTRL-G LENYOMASAVAL TORTENIK.]
 PR [] PR []
 PR [HA ELOLVASTAD NYOMD LE A]
 PR [<RETURN> GOMBOT!]
 PR RQ
 KEZD
END

TO KEZD
 DRAW
 RAJZOL
END

TO RAJZOL
 RAJZ JOYSTICK 1
 IF JOYBUTTON.1 PD ELSE PU
 IF RQ? PUN RQ
 RAJZOL
END

TO RAJZ :IR
 IF :IR < 0 STOP
 SETH :IR * 45 FD 1
END
```

A potméter jeleit kezelő utasítások:

PADDLE $s_1 \rightarrow s_2$
$s_1 \in [\emptyset, 3]$ egész
s_2 : [0, 255] közé eső egész szám, az s_1 kijelölésű potméter értékét adja meg.

PADDLEBUTTON $s \rightarrow t$
$s \in [0, 3]$ egész
t: TRUE, ha az *s* kijelölésű potméter gombja le lett nyomva,
különben FALSE.

2.6.3. Kimeneti egységek jeleit kezelő utasítások

PRINTER
Aktivizálja a sornyomtatót. A képernyőn ezentúl kiíródó szövegeket a sornyomtatóra is kinyomtatja.

NOPRINTER
Nem engedélyezi a sornyomtató használatát.

Kiíró utasítások:

FPRINT, PRINT, PRINT1, .CTYO (l. Írás a szövegmezőre).

PRINTOUT
(PO)
Az utoljára szerkesztett eljárást kilistázza.

PRINTOUT ALL
(PO ALL)
A munkaterület minden eljárását és változó nevét a hozzájuk tartozó tartalommal együtt kilistázza.

PRINTOUT NAMES
(PO NAMES)
A munkaterület minden változó nevét a hozzájuk tartozó tartalommal együtt kilistázza.

PRINTOUT PROCEDURES
(PO PROCEDURES)
A munkaterület minden eljárását kilistázza.

PRINTOUT TITLES
(POTS)
A munkaterület minden eljárásának nevét és a hozzájuk tartozó paramétereket kilistázza.

PRINTOUT *jl*
(PO)
j: '' nélküli jelsorozat
A megadott *j* nevű, ill. az *l* listában felsorolt nevű eljárásokat kilistázza.

2.7. Definiálás, javítás és törlés

2.7.1. Eljárásokra vonatkozó utasítások

DEFINE *jl*
l minden eleme maga is lista. Az első allista az eljárás paramétereit tartalmazza, ha nincs, akkor [ ]. A többi allista tartalma az egyes utasítássorok. Egy allista legfeljebb 255 jelből állhat. Az END utasítást nem kell és nem is szabad itt felsorolni.
Létrejön a *j* nevű eljárás az *l*-ben definiált tartalommal.
Az eljárások definiálását általánosan a TO utasítással hozzuk inkább létre a szerkesztő segítségével.

EDIT
(ED)
Az utoljára szerkesztett eljárás javítását teszi lehetővé szerkesztő állapotban.

EDIT ALL
(ED ALL)
A munkaterület minden eljárásának és a változók tartalmának javítását teszi lehetővé szerkesztő állapotban.

EDIT PROCEDURES
(ED PROCEDURES)
A munkaterület minden eljárásának javítását teszi lehetővé szerkesztő állapotban.

EDIT *jl*
(ED)
j: '' jel nélküli jelsorozat.
A megadott *j*-nevű, ill. az *l* listában felsorolt nevű eljárások javítását teszi lehetővé szerkesztő állapotban.

END

Az eljárás végét jelzi. Csak akkor kell beírni egy szerkesztett eljárás végére, ha rögtön utána egy másik eljárást is létrehozunk anélkül, hogy kilépnénk a szerkesztőből. A szerkesztőből a ⟨RUN/STOP⟩ vagy a ⟨CTRL⟩-C billentyűk lenyomásával lehet kilépni, amely hatására az eljárás végére automatikusan END utasítás kerül.

ERASE ALL

(ER ALL)

A munkaterület minden eljárását és változóit törli.

ERASE PROCEDURES

(ER PROCEDURES)

A munkaterület minden eljárását törli.

ERASE *jl*

(ER)

j: " jel nélküli jelsorozat

A megadott *j*-nevű, ill. az *l* listában felsorolt nevű eljárásokat törli a munkaterületről.

TEXT *j* → *l*

l: lista tartalma a *j* nevű eljárás tartalma. Minden eleme egy allista; első a paramétereket tartalmazza, majd az utána következők sorra az eljárás utasítássorait tartalmazzák.

TO *j* paraméterek

j: " jel nélküli jelsoroza

A *j* nevű eljárás definíciojának kezdetét jelzi. A paraméterek az eljárás lokális paraméterei lesznek, azaz csak akkor léteznek, ha az eljárás aktív. Az eljáráson kívül nincs értelmezve, így elrontani sem lehet az értékét.

A munkaterület eljárásait, neveit kilistáztathatjuk a PRINTOUT, PRINTOUT ALL, PRINTOUT PROCEDURES, PRINTOUT TITLES utasításokkal (l. kiíró utasítások).

Ha szerkesztés közben egy eljárás nevét megváltoztatjuk, úgy a szerkesztőből való kilépéskor két eljárás képződik: egyik az eredeti névvel, a másik az új névvel, de tartalmuk azonos.

2.7.2. Változókra vonatkozó utasítások

Az eljárásokat érintő utasítások közül az EDIT ALL és az ERASE ALL a változókra is hatással van.

EDIT NAMES
(ED NAMES)
A munkaterület minden változójához tartozó tartalom javítását teszi lehetővé szerkesztő állapotban.

ERASE NAMES
(ER NAMES)
A munkaterület minden változóját törli.

ERNAME *j*
A *j* nevű változót törli.

LOCAL *j*
A *j* nevű változót a magába foglaló eljárás számára lokálissá teszi anélkül, hogy paramétere lenne az eljárásnak. Csak akkor létezik a változó, ha az eljárás aktív; az eljáráson kívül nincs értelmezve, így elrontani sem lehet az értékét. Értéket a MAKE utasítással lehet nekik adni.

MAKE *j sjlt*

A *j* nevű változó tartalma *sjlt* lesz. Ezzel az utasítással az értéket kapó változó globális lesz, hacsak a LOCAL utasítással másképpen nem rendelkeztünk. A globális változókra bárhonnan hivatkozhatunk (még parancs szinten is), így értéküket is módosíthatjuk.

THING *j* → *sjlt*
sjlt: az *sj* nevű változó tartalma. Hatása megegyezik a változó elejére írott :jellel, indirekt hivatkozás esetén hasznos lehet.

Például:
MAKE "NÖVÉNY "VIRÁG
MAKE "VIRÁG "LILIOM,

így PRINT THING :NÖVÉNY eredménye

LILIOM,

mivel a NÖVÉNY nevű változó tartalma VIRÁG, amely nevű változó tartalma viszont LILIOM.

THING? *sj* → *t*
t: TRUE, ha az *sj* nevű változónak van tartalma (érvényességi tartományától is függ),
különben FALSE.

2.8. Lemezkezelés

BLOAD *j*
j: a fájl teljes neve (név. típus)*
A *j* nevű gépi kódú – fájlt bemásolja a tárba a kimentéskor megjelölt címre.

BSAVE *j s1 s2*
j: a fájl teljes neve (név. típus)
s1: a gépi kódú program kezdőcíme a tárban
s2: a gépi kódú program végcíme a tárban
$s1 < s2$

Az [*s1*, *s2*] címintervallumban levő gépi kódú programot kimásolja lemezre *j* gépi kódú – fájlnévvel.

CATALOG
Kilistázza a lemezen levő fájlok nevét.

DOS *l*
l tartalmát mint Commodore DOS utasítássorozatot hajtja végre. A műveletek alakja:
N∅: lemeznév, kétjegyű szám
(lemezformattálás)
C∅: új fájlnév = ∅: régi fájlnév
(fájlmásolás)
R∅: új fájlnév = régi fájlnév
(fájlátnevezés)
S∅: fájlnév
(fájltörlés)

ERASEFILE *j*
A *j* nevű szövegfájlt törli a lemezről.

ERASEPICT *j*
A *j* nevű kép-fájlt törli a lemezről.

READ *j*
A *j* nevű lemezes szövegfájlt a munkaterületre másolja. Ha a fájlban szereplő eljárásnevek közül valamelyik megegyezik egy a már bentlevő eljárás nevével, úgy azt felülírja és a régi elvész. Ha a fájlban volt STARTUP változó, úgy a másolás befejeztével annak tartalmát végrehajtja (l. STARTUP).
(A rajzmezőt törli).

*A fájlnév mindig egy szó.

READPICT *j*
Áttér grafikus módba és a *j* nevű lemezes képfájlt a rajzmezőre másolja.

SAVE *j*
Az egész munkaterület tartalmát *j* névvel kimásolja a lemezre mint szövegfájl. (A rajzmezőt törli.)

(SAVE *j l***)**
A munkaterület *l* listában felsorolt eljárásait *j* névvel kimásolja a lemezre mint szövegfájl. (A rajzmezőt törli.)

SAVEPICT *j*
A rajzmezőn levő képet *j* névvel képfájlba másolja a lemezre. A „sprite"-okat nem, így a Teknőc sem lesz rajta a képen.

SETDISK *s*
$s \in [0, 9]$ egész.
Aktivizálja az *s* számú lemezmeghajtót.
Alapállapot: 8.

2.9. Sprite-kezelés

A Commodore 64 lehetőséget ad nyolc különálló alakzat, „sprite" tervezésére, és irányítására. A sprite olyan egység, amely egy definiált alakzatot hordozhat. A sprite-ok 0-tól 7-ig vannak számozva, ugyanúgy az alakzatok is. Alapállapotban minden „sprite" a számával megegyező alakzatot hordozza, amelyet mozgáskor magával visz. A sprite-ok irányítása a Teknőc-utasításokkal történik, hiszen a Teknőc maga is egy sprite, a 0.

BLOAD *j*.**SHAPES**
A *j* nevű alakzatfájlt betölti a tárba a kimentéskor megjelölt címre.

***SETSHAPE** *s*
$s \in [0, 7]$ egész.
Az aktuális „sprite" felveszi az *s* számú alakzatot. Alapállapotban minden sprite a számával megegyező számú alakzatot hordozza és csak a 0. sprite veheti fel a forgó Teknőc-alakzatot.

***SHAPE** → *s*
s: [0, 7] közé eső egész szám, az aktuális sprite alakzatának száma.

TELL *s*
$s \in [0, 7]$ egész.
Aktivizálja az *s*. számú „sprite"-ot. Ezután minden Teknőc-utasítás ennek a „sprite"-nak szól, amíg másértelmű TELL utasítást nem adunk. Alapállapot-

ban a 0. sprite látható és tollal lent van, 1.–7. sprite viszont nem látható és tollal fent van.

*WHO → *s*
s: [0, 7] közé eső egész szám, az aktuális sprite száma.
Alapállapot: 0.

*.SPRINT *s*
s = 64 * (48 + *a*), ahol *a* = alakzat szám.
Kilistázza a szövegmezőre az *a* számú alakzat nagyított pontjait.
(Az utasítás utáni rész a sor végéig nem hajtódik végre).

2.10. Speciális utasítások

.CONTENTS → *l*
A munkaterület összes aktív nevét kilistázza (változónevek, eljárásnevek, jelsorozatok ... stb.)

.GCOLL
„Szemétgyűjtés". A rendszer szabad „csomóit" összegyűjti. Egy „csomó" a rendszer legkisebb felhasználható egysége (belső használatra). Csak rendkívüli esetben érdemes használni az utasítást.
A rendszer szükség esetén maga is végez „szemétgyűjtés"-t.

.NODES → *s*
s: a rendszerben levő szabad „csomók" száma. .OPTION *utasítás* a *tulajdonság módja, tulajdonság* ∈[0, 2] egész szám az utasítás tulajdonságát jellemzi. *mód* ∈[0, 7] egész szám, a módosítás módját jellemzi.
Módosítja az utasítás adott tulajdonságának hatását adott módon.
Használata a Commodore 64 belső „világának" jobb ismeretét igényli.

Gépi kódú programozás:

.CALL *s*1 *s*2
A vezérlés áttér a felhasználó által definiált *s*1 címen levő gépi kódú eljárásra, amely az *s*2-t paraméterének tekintheti. Az eljárás végrehajtása után a vezérlés visszatér.

.DEPOSIT *s*1 *s*2
Az *s*1 (decimális érték) kezdetű tár címre lerakja az *s*2 decimális számot, mint hexadecimális érték.

.EXAMINE *s*1 → *s*2
*s*2: az *s*1 (decimális érték) című tár területen levő byte tartalma.

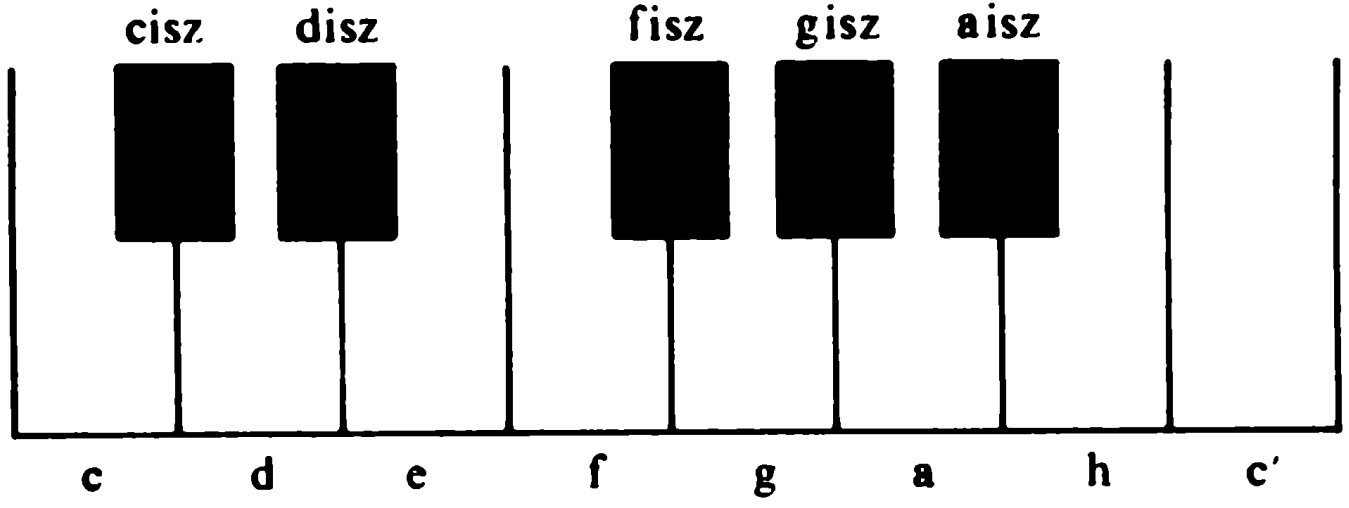
Ezeknek az utasításoknak a használata a Commodore 64 memóriatérképét és a gépi kódú programozás mélyebb ismeretét igénylik.
A következő példa mutatja, hogyan tudunk segítségükkel hangot kicsikarni a gépből.

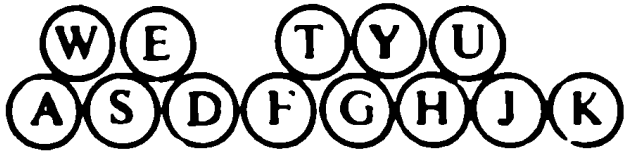
```
TO ZONGORA
 PIP 0 ERTEK RC 15 60
 ZONGORA
END

TO PIP :MAG :FREK :ERO :IDO
 HANG :MAG :FREK :ERO
 HANG.KEZD :MAG 17
 VARJ :IDO
 HANG.VEG :MAG
END

TO HANG :MAG :FREK :ERO
 .DEPOSIT 54296 :ERO
 .DEPOSIT 54272 + :MAG * 7 REMAINDER :FREK 256
 .DEPOSIT 54273 + :MAG * 7 INTEGER :FREK / 256
 .DEPOSIT 54277 + :MAG * 7 0
 .DEPOSIT 54278 + :MAG * 7 250
END

TO HANG.KEZD :MAG :HLF
 .DEPOSIT 54276 + :MAG * 7  HLF
END

TO HANG.VEG :MAG
 HANG.KEZD :MAG 0
END

TO VARJ :IDO
 REPEAT 5 * :IDO [ ]
END

TO ERTEK :S
 IF :S = "A OP 4455 STOP
 IF :S = "S OP 5001 STOP
 IF :S = "D OP 5613 STOP
 IF :S = "F OP 5947 STOP
 IF :S = "G OP 6676 STOP
 IF :S = "H OP 7493 STOP
 IF :S = "J OP 8411 STOP
 IF :S = "K OP 8911 STOP
 IF :S = "W OP 4720 STOP
 IF :S = "E OP 5298 STOP
 IF :S = "T OP 6301 STOP
 IF :S = "Y OP 7072 STOP
 IF :S = "U OP 7939 STOP
 OP 0
END
```

Az eljárások beírása után adjuk ki a következő utasítást:

ZONGORA

Ezekután a meghatározott gombok lenyomása a nekik megfeleltetett hangokat adják ki:





2.11. Hibaüzenetek

A hibaüzenetek verziónként kissé különbözhetnek, de mindenképpen segítséget nyújtanak a hiba lokalizálásában.

Állandó szövegű hibaüzenetek:

CAN'T DIVIDE BY ZERO
(nem tudok nullával osztani).

DISK ERROR
(lemezhiba, nem megfelelő lemezhasználat).

END SHOULD BE USED ONLY IN THE EDITOR
(END csak az eljárásdefiníció végén szerepelhet).

ELSE IS OUT OF PLACE
(ELSE rossz helyen van, csak IF...THEN...ELSE szerkezettel lehet használni).

FILE NOT FOUND
(a fájlt nem találja a lemezen).

LINE GIVEN TO { DEFINE / REPEAT / RUN } TOO LONG
(túl hosszú a megnevezett utasítás paramétere).

MISSING INPUTS INSIDE ( ) 'S
(a ( ) jelek között hiányzik paraméter).

NO STORAGE LEFT!
(nincs több szabad hely – töröljük az eljárások közül, ha tudunk).

NUMBER TOO LARGE OR TOO SMALL
(a szám túl nagy vagy túl kicsi).

PRINTER NOT READY
(a nyomtató nem áll készen a munkára).

PROCEDURE NESTING TOO DEEP
(az eljárások túl mélyen vannak egymásba ágyazva).

THE : IS OUT OF PLACE A T jelsorozat
(a : jel rossz helyen van a kiírt jelsorozatnál).

THE DISK IS FULL
(a lemez tele van).

THE DISK IS WRITE PROTECTED
(a lemez írásvédve van).

{THEN / ELSE} IS OUT OF PLACE
({THEN / ELSE} rossz helyen van).

THERE IS NO DISK IN THE DISK DRIVE
(nincs lemez a lemezegységben).

THERE IS NO LABEL jelsorozat
(nincs jelsorozat nevű címke).

THERE IS NO NAME jelsorozat
(nincs jelsorozat nevű változó).

THERE IS NO PROCEDURE NAMED jelsorozat
(nincs jelsorozat nevű eljárás).

THERE IS NOTHING TO SAVE
(nincs mit kimásolni a lemezre).

TOO FEW ITEMS
(túl kevés paramétert adtunk).

TOO MANY CHARACTERS TO eljárás
(eljárás elvégzéséhez túl sok karaktert adtunk meg).

TOO MANY PROCEEDURE INPUTS
(túl sok eljárásparaméter van).

TOO MUCH INSIDE PARANTHESES
(túl sok paraméter van a zárójelek között).

TURTLE OUT OF BOUNDS
(a Teknőc letéra rajzmezőről).

YOU DON'T SAY WHAT TO DO WITH adat
(nem adtuk meg mit csináljon az adattal).

{TITLES / PROCEDURES / NAMES / ALL} SHOULD BE AN INPUT ONLY TO PRINTOUT, ERASE OR EDIT
(ezek a nevek csak a felsorolt utasítások paraméterei lehetnek).

Utasítás NEEDS MORE INPUTS
(az utasítás több paramétert vár).

Utasítás DIDN'T OUTPUT
(az utasítás nem adott eredményt).

Műveletjel NEEDS SOMETHING BEFORE IT
(a műveleti jel egy paramétert igényel előtte).

Utasítás DOESN'T LIKE paraméter AS INPUT
(az utasítás nem ilyen paramétert vár).

Néha a következő kifejezések követik
IT EXPECTS TRUE OR FALSE
(TRUE vagy FALSE a várt paraméter)

vagy IT EXPECTS AN UNQUOTED WORD
(" jel nélküli jelsorozatot vár).

Utasítás IS A LOGO PRIMITIVE
(az utasításnév LOGO alaputasítás – ilyen névvel eljárást nem lehet elnevezni).

{ OUTPUT / STOP / LOCAL / GO } SHOULD BE USED INSIDE A PROCEDURE

(ezek az utasítások csak eljárásdefinícióban használhatók).

Az eljárások végrehajtása közben észlelt hibaüzenet a hiba helyét is megmutatják.

üzenet IN LINE
utasítás sor
AT LEVEL szint szám OF eljárásnév.

3. A Snail Logo Sinclair ZX Spectrumra

3.1. A Snail Logo használata

3.1.1. Bevezetés

E fejezetben gyakorlati példák leírásával mutatjuk be a Snail Logo használatát. Mivel ez az első ismerkedés a nyelvvel, ezért a Snail Logoban rejlő lehetőségekről csak kevés példát adunk. Áttanulmányozva a leírást, már képes lesz az Olvasó a bemutatott példákat bővíteni és általánosítani, valamint saját elgondolásait megvalósítani.

Ha valahol megakad, lapozza fel a kulcsszavak leírását, ahol mindenre talál magyarázatot.

Mielőtt a példákkal kezdenénk, néhány fontos kifejezést kell megismerni. Ezek:

– utasítás, program, eljárás.

Az utasítás:

A Snail Logo utasítás egy parancsból áll, néha a parancsszó után numerikus számérték vagy változónév áll. Az utasítások vagy közvetlenül a Csigára hatnak, vagy más utasítások végrehajtását módosítják. Az összes utasítást az 1.3. szakaszban ismertetjük, működésük magyarázatával együtt. Általános elvek a következők:

a) A Csigának mindig van egy adott pozíciója és iránya. A helyzetet a FORWARD, BACKWARD utasításokkal és a lépésszámmal, az irányt a RIGHT, LEFT utasításokkal és a fokokkal változtathatja meg. (Megjegyezzük, hogy a CENTRE, POSITION, NORTH és PNORTH szintén változásokat okoz.)

b) A Csiga vagy látható, vagy láthatatlan a SNAIL és NSNAIL utasításoknak megfelelően.

c) A Csiga nyoma is vagy látható, vagy nem, annak megfelelően, hogy a toll lent (DOWN) vagy fent (UP) van. Ha a nyomvonal nem látszik, akkor az aktuális pozícióhoz tartozó nyom az OUTPUT utasítás segítségével megnézhető. A COLOUR utasítás a nyomvonal színét definiálja.

d) REPEAT és PROCEDURE utasítások más utasításokat befolyásolnak, ezeket később ismertetjük.

A program:

Egy program Snail Logo utasítások listája, amelyek programfutáskor egymás után végrehajtódnak. A REPEAT és PFINISH utasítások közé írt utasítások annyiszor hajtódnak végre, ahányszor a REPEAT utasítással deklaráljuk. A programot mindig END utasítással zárjuk. A változókat a SET, INCREASE, DECREASE és MAKE utasításokkal kezelhetjük, az eredményt SHOW utasítással kaphatjuk meg. Egy program vagy eljárás bizonyos feltételek mellett IFEND utasítással is befejeződhet.

Az eljárás:

Egy eljárás – a programhoz hasonlóan – utasítások listája. De az eljárás közvetlenül nem futtatható, csak a program. Az eljárásokat programok vagy más eljárások használják. Minden eljárásnak szabadon választható neve van (pl. SQUARE, CIRCLE), amelyek utalhatnak az eljárásban található utasítások működésére. A programok az eljárás nevének megadásával az eljárás összes utasítását használhatják. Ilyen pl. a PROCEDURE SQUARE.

3.1.2. Példák

Ha a következő, 1.1.2.–1.1.8. példák utasításait követi az Olvasó, akkor az eredményt a képernyőn fogja megkapni. Minden esetben érdemes elgondolkodni a Snail Logo utasításai és a kapott grafikus minták közötti összefüggéseken.

Mielőtt rátérne az 1.1.3. szakaszra, nézze meg az 1.3.1. szakaszban felsorolt Snail Logo utasításokat, innen tudhatja meg az egyes utasítások hatását!

A Snail Logo a képernyőn látható "Menu" rendszerrel működik. A menü három csoportban tartalmazza a rendszer által kínált lehetőségeket. Az adott művelet egybetűs kódját beadva (egy esetben ez hárombetűs) a művelet elérhetővé válik. Végrehajtás után a rendszer a menüre tér vissza, és már választható a következő. A műveletekre csak akkor hivatkozhatunk, ha azok a képernyőn ténylegesen ki vannak írva.

Csak az ENTER billentyű megnyomása szükséges ahhoz, hogy az egyik műveletcsoportról áttérhessünk a másikra, és végül vissza a menüre. A programok és eljárások begépelését és a programok szerkesztését mindig az END beírásával fejezzük be, mielőtt ENTER-rel visszatérnénk a menüre, különben a szöveg elvész. (L. még az 1.2.3. szakasz 1. pontját!)

Most pedig ellenőrizze, hogy megértette az elmondottakat, és figyeljen a Z és U billentyűk használatára! Figyelem, gyakran az ENTER-t többször is meg kell nyomni, hogy adott műveletről a menüre térjen vissza!

Talán bosszantónak találja a különböző ellenőrzéseket és hibaüzeneteket. De hamarosan megszokja az egyszerű szabályokat, és ez ellenőrzések a gyerekeknek különösen fontosak. Amennyire lehetséges, győződjön meg a Snail Logo program épségéről, mielőtt futtatná, így az esetleges programhiba miatt nem kap hibás vagy zavaró eredményeket. Hacsak programból nem változtatja meg, a Csiga a képernyő közepéről indul északi irányban, tollal le, és a Csiga nem látható.

Bátran nyomja a gombokat! Nem árthat sem a gépnek, sem a Snail Logo programnak, bár ha gyakran nyomogatja a BREAK billentyűt, a Snail Logo megáll, és ekkor újra be kell tölteni a szalagról.

Az első példák részletesen ismertetik a lehetőségeket, míg a későbbiek a részleteket már nem ismétlik. Ha szükséges, ismételten nézze át az első példákat.

Végül, ha egy adott ponton nem tudja, mit kellene tennie, mindig az ENTER billentyűt próbálja megnyomni.

3.1.3. Első lépések

A műveletekhez tartozó betűket be kell adni, ami azt jelenti, hogy az adott billentyű megnyomása után még az ENTER-t is meg kell nyomni. Amíg az ENTER-t nem nyomja meg, nem történik semmi.

Ha nincs nyomtatója, akkor a C beadását hagyja figyelmen kívül. Csak az ENTER gombot nyomja meg.

1. Töltse be az 1.2.2. pont alapján a Snail Logo rendszert! A betöltés végén menü jelenik meg, és a cím „Snail Logo", amelyet néhány pont és a jelképes Csiga követ.

2. A Caps Lock automatikusan beállítódik, mivel a Snail Logo rendszer csak nagybetűket használ.

3. A menü megjelenésekor nyomja meg az R gombot (RUN művelet), ezzel a szalagon mellékelt demonstrációs program futtatható. Bár ez nem tipikus Snail Logo program, számos technikát és a Csigaszimbólum használatát bemutatja.

4. Nyomja meg a C gombot, ha kinyomtatott programlistát szeretne! Különben az ENTER-t, és akkor visszajut a menühöz.

5. Nyomja meg az L billentyűt, ami a FLOWER programot listázza!

6. Nyomja meg most az ENTER-t, megjelenik a (P)ROC? üzenet. Hagyja figyelmen kívül!

7. Ismételten nyomja meg az ENTER-t, most a (C)OPY üzenet jeleni meg! Nyomja meg a C gombot, és ezzel a programlistát a nyomtatóra irányítia. Listázás után ismét a menüre ugrik vissza a rendszer.

A lista az utasítások rövidített alakját tartalmazza, minden utasítás nevének első két betűjét. Ellenőrizze az 1.3.1. pont alapján az utasítások teljes nevét! A program teljes megértése ebben a stádiumban még nem szükséges.

8. Visszatérve a menüre, nyomja meg a K gomot, majd eljárásnévként gépelje be, hogy "STEM"! Vessen egy pillantást az eljáráslistára! Ha a listát a nyomtatón szeretné látni, akkor a (C)OPY üzenetnél nyomja meg a C gombot! Különben ENTER.

9. A 8. pontban leírt módszerrel ellenőrizze a LEAF és BLOOM eljárásokat! Az ENTER megnyomásával térhet vissza mindig a menüre.

3.1.4. Következő lépések

Itt az ideje, hogy olyan utasításokat gépeljen be és próbáljon ki, amelyek már Snail Logo programot alkotnak. Az utasítás beadásához a teljes utasítást be kell gépelni, pl. FORWARD 29, ilyenkor a szöveg a képernyő alján jelenik meg. Ebben a stádiumban a DELETE gomb a Basic-hez hasonlóan használható. Ha készen vagyunk az utasítással, az ENTER gombot nyomjuk meg.

1. A menüre visszatérve írja be, EPR! (Ezzel most átírta a FLOWER programot.) A rendszer kész új programutasítások elfogadására.

2. Gépelje be, hogy 123ABC! Az eredmény hibaüzenet lesz. (A különböző hibaüzeneteket a 3.4. szakasz tartalmazza.) Nyomja meg az ENTER-t, ezzel figyelmen kívül hagyja az üzenetet! (Az EN a képernyőn ENTER-t jelent!)

3. Írja be FORWARD 6 (fontos az egy betűköz!), majd END! Ez már egy elfogadott program. Az ENTER megnyomásával a menüre tér vissza.

4. Nyomja meg most az R gombot, ezzel a programot futtatja, közben figyelje a képernyőt! Ellenőrizze a Csiga által megtett hat lépést előre! Az ENTER megnyomásával tér vissza a menüre.

5. Írja be, hogy EPR, majd a következő programot! (Használja az ENTER gombot minden utasítás begépelése után!)

```
REPEAT 6
BACKWARD 1
RFINISH
END
```

6. Visszatérve a menüre, futtassa a programot az R megnyomásával! Figyelje meg a REPEAT és BACKWARD utasítások hatását!

7. Az L megnyomásával listázza a programot! Nyomja meg az ENTER-t, és a megjelenő (P)ROC? üzenet után a P billentyűt, ezzel a programot eljárássá alakította. A rendszer meg fogja kérdezni az eljárás nevét. Írja be, SIXBAC, majd az eljárás sorszámát, ami most legyen 1!

8. Gépelje be a következő programot (EPR), ami a SIXBAC eljárást használja, és futtassa is le!

```
REPEAT 2
PROCEDURE SIXBAC
RFINISH
END
```

Figyelje meg az eljárás kétszeri használatát!

9. Gépelje be és próbálja ki a következő, rövidített utasításokkal megadott programokat! Használja a 3. fejezetet az utasítások ellenőrzésére! A programok beírása során figyelje meg a Z és U gombok hatását!

a)
```
SN
RI 90
FO 8
END
```

b)
```
PO 10. 10
RE 4
RI 90
FO 8
RF
END
```

c)
```
RE 9
FO 15
RI 160
RF
END
```

10. Visszatérve a menüre, a P billentyű segítségével nevezze el az *a)*-t SQUARE eljárásnak, adjon sorszámot is neki! Írjon olyan eljárást, amely 6 lépés oldalhosszúságú négyzetet rajzol! (Nézze meg a 9*b)* példát!) Írjon és próbáljon ki egy olyan programot, amely a SQUARE eljárást használja! (Nézze meg a 8-as pontot, most persze ismétlésre nincs szükség!)

11. A menüre visszatérve nyomja meg az N gombot!

12. Adja be és futtassa a következő programot:

```
RE 3
PR SQUARE
RI 120
RF
END
```

13. Visszatérve a menüre, nyomja meg az E gombot, ezzel szerkeszteni lehet a beírt programot! Adja be a hármas sor számát (3), és nyomja meg az R gombot a cseréhez! Végül gépelje be, hogy RI 90 és END! Ezzel az RI 120 utasítást RI 90-re cserélte ki! Mikor az átírt programot listázza, nyomja meg a Z gombot, ezzel befejezte a szerkesztést! Futtassa az átírt programot!

14. Listázza a programot, és alakítsa át eljárássá, legyen a neve BLOCKS, sorszáma pedig 3! Gépelje be és próbálja ki a következő programot!

```
PO 20, 20
PR BLOCKS
PO 40, 20
PR BLOCKS
END
```

Figyelje meg az eljárások hierarchiáját! Minden eljárás más eljárást használ. Hogyan lehetne a két BLOCKS-ot azonosra átírni?

3.1.5. Változók

A következő példák a változók használatát mutatják be. Ez jelentősen növeli a Snail Logo nyelv hatékonyságát.

A változó olyan számérték, amelynek neve van. A változóra nevével hivatkozunk, az értékének megfelelő szám tetszőleges lehet, sőt meg is változtatható, hiszen ezért nevezik változónak.

A Snail Logonak nyolc különböző változója van. Ezek neve a következő: A, B, C, D, E, F, G, H. Értékük bármely pozitív vagy negatív szám lehet. Közvetlen értékként csak pozitív egész adható, de aritmetikai műveletek hatására negatív vagy törtszámok is lehetnek.

A változók különböző módon viselkednek. Két különböző típusuk van, a globális és a lokális.

Globális változók: E, F, G, H.

A globális változót a Snail Logo rendszer bármely programja, eljárása használhatja. Például ha az ALMA eljárás az E változónak 17-et ad értékül, akkor ezt a NARANCS eljárás kiolvashatja vagy megváltoztathatja. Csak egy E, F, G és H változó létezhet.

Lokális változók: A, B, C, D.

A lokális változóknak számos példánya lehetséges. Minden programnak és eljárásnak saját lokális változói vannak. Például ha egy program A-nak 3-at ad értékül, akkor ez az ALMA eljárásban nem vehető észre. Az ALMA saját A változójának pl. adhat 24-et értékül, ez egyáltalán nem zavarja a program A változójának hármas értékét. A NARANCS eljárás még akár egy harmadik értéket is adhat A-nak, ezen A értékek egymástól függetlenül léteznek.

Egy mód van csak arra, hogy egy program vagy eljárás egy másik eljárás A, B és C (de nem D) változójának értéket adhasson: a PROCEDURE utasítás. Például a PR ALMA, 9 nemcsak az ALMA eljárást hívja meg, hanem az ALMA eljárás A lokális változójának 9-et ad értékül. A PROCEDURE utasítás nincs hatással a hívó program vagy az eljárás változójára. E lehetőség használatát az 1.6. Eljárások és paraméterek c. szakaszban részletezzük.

A „D" változó

Alapvetően a D egy közönséges lokális változó, de úgy is kell használni. De ezt használja a Snail Logo rendszer a Csiga irányának tárolására az egyes programokba, ill. eljárásokba való belépéskor. Ez teszi lehetővé az RNORTH utasítás működését, amely az eljárásnak megfelelően fordítja el az irányt északtól. Ha D értékét valamilyen utasítással megváltoztatjuk, akkor az RNORTH nem fog helyesen működni, bár ez a hatás komolyabb programoknál jól használható.

A változók ellenőrzése és használata

Változókat számos utasításban is használhatjuk, mint pl. FORWARD A, REPEAT F. Persze ekkor már előre megadott értékük van. A Snail Logo kezdőértékül minden változónak nullát ad. Más érték pl. a már említett PROCEDURE utasítással adható, vagy pl. a következő módokon:

SET	direkt értékadás	SET C, 19
INSCEASE	az aktuális értékhez ad egy másikat	INC F, 3
DECREASE	az aktuális értékből kivon egy másikat	DEC G, 1
MAKE	aritmetikai műveleteket tesz lehetővé	MAKE A=B+C
		MAKE C=F / G
		MAKE D=H*A
		MAKE E=D–A.

Egy adott változó futás közbeni értékét a SHOW utasítással írhatjuk ki, pl. SHOW C.

Jegyezzük meg, hogy a MAKE utasításban a szorzás jele csillag (*), hasonlóan a Basic-hez.

1. Gépelje be és futtassa a következő programokat!

a)
```
SET A,10
RE 6
INC A,20
FO 6
CE
SH A
RI A
RF
END
```

b)
```
SET A,4
SET B,5
MA G=A+B
MA D=A/B
SH G
SH D
END
```

2. Írja be a következő, TAR nevű eljárást (sorszáma legyen 5), majd gépelje be a programot és próbálja ki!

```
PROGRAM:
SE H,5
RE 4
FO 5
PR TAR
CE
RI 90
RF
END
```

```
PROCEDURE TAR:
RI 90
DE H,1
FO H
RN
END
```

Ez a program egyrészt az RN utasítás használatát (próbálja meg NORTH-ra kicserélni), másrészt a H változó globális természetét szemlélteti.

3.1.6. Eljárások és paraméterek

A következő példák az eljárások paraméterhasználatát mutatják be. Mielőtt továbbmenne az Olvasó, ajánljuk, hogy nézze át a 3.1. szakaszt a PROCEDURE utasítással adhatunk értéket. Például:

```
PROCEDURE ALMA, 310, 90.
```

Az eljárás neve után szereplő első szám az A, a második a B, a harmadik pedig a C változónak ad értéket. Ezt a technikát paraméterátadásnak nevezik. Ezért a program ugyanazt az eljárást többször hívhatja, és különböző eredményeket kaphat a különböző paraméterértékek átadásával. Például három, eltérő méretű négyzetet rajzolhat a három utasítással:

```
PROCEDURE SQUARE,3 PROCEDURE SQUARE,5
PROCEDURE SQUARE,9.
```

Jegyezze meg, ha a PROCEDURE utasítás REPEAT hurokban található, akkor a paraméterek csak az első hívásnál adódnak át! Így a paraméterek nem állítódnak vissza újabb hívások esetén.

1. A P gomb megnyomásával nevezze el a következő eljárást CIRCLE-nek, és legyen a sorszáma 4!

```
RE 8
FO A
RI 45
RF
END
```

Gépelje be és próbálja ki a következő programot! Figyelje meg, az eljárásnak átadott paraméter értéke 2, és ezt tekinti az eljárás az A változó értékének.

```
RI 20
RE 4
PR CIRCLE, 2
LE 90
RF
END
```

2. Cserélje ki a program 3. sorát PR CIRCLE,4-re, és ismét futtassa le! Majd szúrja be a PR CIRCLE,4 utasítás után a PR CIRCLE,2-t (használja a 3 vagy 4 sorszámokat)! Próbálja ki a programot!

3. Írja meg a POLY eljárást (válasszon most már maga sorszámot neki) és egy programot, amely különböző A értékekkel hívja!

```
PROCEDURE POLY                PROGRAM
MA B=E/A                      SET E,360
```

```
RE A
FO 6
LE B
RF
END
```

```
PR POLY,3
PR POLY,5
PR POLY,7
END
```

Az A * B = 360 összefüggés használata is ezt az eredményt adja. Próbáljon ki egyéb összefüggéseket is!

Ha van két perc ideje, ezt a programot érdemes megvárni. Használja a POLY eljárást! Az eredmény ugyanaz lesz, és gyorsabban fog futni, ha egy programként van megírva.

4. Próbálja ki ezt:

```
PROCEDURE SPIRAL
FO C
IN C,1
RI 90
END
```

```
PROGRAM
RE 50
PR SPIRAL,1,1,1
RF
END
```

Figyelje meg, hogy az A-nak és B-nek adott értékre csak a C megadása miatt van szükség. Ezt a programot a Z billentyű lenyomásával le tudja állítani befejezés előtt.

5. Írjon egy TRISPI nevű eljárást és egy programot, ami egy háromszögletű spirális készítésére alkalmas. Használja ezt!

6. Próbálja ezt:

```
PROCEDURE SPITRI
FO A
IN A,1
RI 120
END
```

```
PROGRAM
RE 50
PR SPITRI,1
RI 5
RF
END
```

7. Próbálja ki ezt:

```
PROCEDURE GALAXY
FO A
IN A,1
RI 149
END
```

```
PROGRAM
UP
RE 50
OU
PR GALAXY,1
RF
END
```

Kísérletezzen különböző szögekkel a GALAXY eljárásban!

8. Végül figyelje meg, hogy a következő egyszerű eljárás és program milyen különböző alakokra használható csupán a paraméterek megváltoztatásával!

```
PROCEDURE FLEXI
RE A
FO B
RI C
RF
END
```

```
PROGRAM
PR FLEXI,3,4,720.
END
```

3.1.7. Rekurzív eljárások

Már megmutattuk, hogy hierarchikus, egymásba ágyazott eljárások írhatók. Egy program hívhatja a HOUSE eljárást, amely hívhatja a DOOR nevűt, ami esetleg hívja a KNOB eljárást. Az eljárások azonban saját magukat is hívhatják, ezt nevezik rekurziónak.

1. Próbálja ki ezt:

```
PROCEDURE FRED
FO 1
PR FRED
END
```

```
PROGRAM
PR FRED
END
```

2. Változók használhatók és változtathatók. Pl.:

```
PROCEDURE JOE
FO A
IN A,1
PR JOE,1
END
```

```
PROGRAM
PR JOE,1
END
```

Ha Bounce be van állítva és Z nincs lenyomva, mindkét előbbi példaprogram addig fog futni, ameddig a rekurziós verem ki nem ürül, amikor is az END OF RECURSION STACK hibaüzenet jelenik meg. Ennek elkerülésére hasznos az IFEND utasítás.

3.

```
PROCEDURE JIM
SET B,5
FO A
IN A,1
IFEND A>B
PR JIM,1
END
```

```
PROGRAM
PR JIM,1
END
```

Az IFEND utasítás feltételei használhatnak lokális vagy globális változókat. Az utóbbiakat más eljárások vagy programok is beállíthatják. (Hogyan gyor-

sítható fel egy kicsit a 3. példa egy globális változó használatával?) Figyelje meg, hogy az IFEND nemcsak rekurzív eljárásokban használható, hanem bármilyen szituációban. Egy REPEAT hurokban használva azonban leállítja a végrehajtást. A 2. és 3. példaprogram futásának gondos vizsgálata megmutatja, hogy mielőtt A 2, 3 stb. értékekre növekszik, két egyszerű lépés történik. Ez fontos dolgot illusztrál. Amikor egy eljárás rekurzív módon magát hívja, akkor a lokális változóinak előzőleg megváltoztatott értéke hatásos. Így az eljárás első hívásakor a program beállíthatja ezeknek a változóknak az értékét, és az eljáráson belüli rekurzív híváskor ezek a változók újradefiniálhatók. A változók legelső értéke azonban csak az első rekurzív hívásig érvényes, ellenkező esetben az eljárásban megváltoztatott értékek minden rekurzív híváskor törlődnének. Ezekután világos, hogy az előbbi példákban miért van a két lépés.

Próbáljon ki különböző értékeket, és SHOW utasításokkal vizsgálja meg a hatást! Használjon kezdőértéket a programbeli hívásnál, hogy elkerülje a kezdeti mozgást, úgy mint a következő példában!

4. Próbájla meg ezt:

```
PROCEDURE REC
RE 8
FO A
RI 45
RF
IN A,1
IF A>H
PR REC,1
END
```

```
PROGRAM
SE 4,5
PR REC,0
END
```

5. Vagy ezt:

```
PROCEDURE SQR
RE 4
FO 4
RI 90
RF
IN A,1
RI 25
PR SQR,1
END
```

```
PROGRAM
PR SQR,1
END
```

3.1.8. Számítások

Az alakzatokkal kapcsolatos egyszerű számítások a következőképpen végezhetők:

1. Kerület

```
PROCEDURE MOV          PROGRAM
MA H=H+A               SE 4,0
FO A                   RE 2
SH H                   PR MOV, 6,9,0
RI B                   PR MOV, 4,9,0
END                    RF
                       END
```

2. Terület

A területek általában a kerületszámításhoz hasonló módon számolhatók, azonban a területet háromszögekre és téglalapokra kell felosztani. Ez hatásosan bemutatható a körvonal megrajzolásával, majd az alakzat felosztásával, miközben a megrajzolt részlet kiszámított területét egy második változóban összegezzük. Itt érdemes alkalmazni a színezést. Különböző technikák használhatók, a gyakorlást az Olvasóra bízzuk.

3.2. A Snail Logo operációs rendszere

3.2.1. Bevezetés

Itt nem közvetlenül a Snail Logo nyelvvel vagy használatával foglalkozunk, hanem a Snail Logo működésének különböző szempontjait mutatjuk be. Ha még nem ismeri egy operációs rendszer szokásos tulajdonságait, az nem jelent problémát. Ezek mind könnyen használható formában adottak, és lehetővé teszik a Snail Logo nyelv rugalmas használatát, azaz:

- a nyelv utasításainak beadását programként;
  a figyelmeztetést téves vagy nemlétező utasítások esetén;
  változtatást, törlést vagy kiegészítést a programban;
  eljárások beírását, egy program eljárássá alakítását vagy egy eljárás törlését;
  egy program vagy eljárás képernyőre listázását, kinyomtatását, valamint a definiált eljárások nevének felsorolását;
- egy program futtatását és a Csiga nyomának képernyőre rajzolását, valamint az eredmény kinyomtatását.

3.2.2. A Snail Logo rendszer betöltése

1. Olvassa el számítógépének kézikönyvét a szalagról való programbetöltéssel kapcsolatosan!
2. Töltse be a szalagot a magnetofonba, majd LOAD " "!
3. Néhány másodperc múlva fel kell tűnnie a címnek. Ha ez nem történik meg, állítsa be a magnetofon kimeneti szintjét és próbálkozzon újra! A teljes betöltés további másfél percet vesz igénybe. A programbetöltés befejeztével a fő funkciók listája, a menü jelenik meg.

3.2.3. Az operációs rendszer lehetőségei

1. Általános szempontok

A menü képernyőjét az 1.2. szakaszban ismertettük. Az operációs rendszer utasításai a menüből érhetők el, és a végrehajtás után a menüre térnek vissza. Az utasítások a menüben felsorolt kódbetű vagy betűk beadásával hívhatók.

Általában az ENTER billentyűt használja az operációs rendszer az utasítások végrehajtására és a menübe való visszatérésre. Egy kivétel van ez alól, amikor programutasítás beadása kezdődött. Ez program esetén (EPR után) vagy eljárás esetén (P után), valamint szerkesztésnél egy utasítás beszúrásakor áll elő. Ha a programutasítás beadása elkezdődött, akkor azt vagy az END utasítással, vagy Z beadásával lehet lezárni a menübe való visszatérés előtt. Ez akkor is érvényes, ha ténylegesen egyetlen programutasítás sem volt beírva.

Nyomja a Z-t a programutasítás beadásának lezárására. Nyomja az U-t, ha törölni akarja az utolsó programutasítást. Az U ismételt beadásával több programutasítás is törölhető. A következőkben az operációs rendszer összes lehetőségét leírjuk. Használatában a gyakorlat teszi a mestert! A leírás általános útmutatásokkal és hibaüzenetekkel van ellátva.

Minden utasítás a Snail Logo részére nagybetűkkel kell, hogy történjen, ezért győződjön meg, hogy a CAPS LOCK be van-e kapcsolva! Ha még nem tette volna, akkor fussa át a programnyelv parancsszavainak referencialistáját a 3. fejezetben, mielőtt a hátralevő részt tanulmányozza.

2. Programbeadás (EPR)

Az operációs rendszernek ez a tulajdonsága lehetővé teszi Snail Logo programutasítások egymás utáni beadását programírás céljából. Minden egyes programutasítást ENTER-rel kell lezárni. Mivel egy új program törli az előzőt, egy hárombetűs kód szolgál a programok megkülönböztetésére. A sorszámozás automatikus. A programutasítások és számok egyszerű elrendezését (az utasítások szintaxisát) a 3. fejezetben mutatjuk be. A programutasítások,

számok és nevek között szóközt kell hagyni, a változókat és a számokat vesszővel kell elválasztani. Ha egy utasításnak rossz a szintaxisa vagy nem értelmezhető, akkor megjelenik a hibaüzenet. Ezeknek a hibaüzeneteknek a listája a 3.4. szakaszban van. Hibaüzenet után ENTER-rel lehet törölni az utasítást. Az utasítások nevének csak az első két betűje kerül megvizsgálásra, ezért a további betűk elhagyhatók, és a rövidített forma használatos. Ez kis gyakorlás után már érezhetően előnyös. (Azonban pl. a FOSSIL 19 és CLOUD parancsok FORWARD 19-nek és CLEAR-nek értelmeződnek!)

Minden program utolsó utasítása (még egy „üres", utasítás nélküli programé is) az END kell, hogy legyen.

A REPEAT hurokban levő utasítások eltolva jelennek meg a listában, így könnyen felismerhetők. Egymásba ágyazott REPEAT hurkok jobban eltolódnak.

A REPEAT és RFINISH utasításoknak a száma egy programban azonos kell, hogy legyen. Ha a számuk nem azonos, és egy END utasítással le akarná zárni a programot, akkor az "RF NOT EQUAL RE" hibaüzenet figyelmezteti, és nem engedi az END-et végrehajtani. ENTER-rel törölheti a hibaüzenetet, és a szükséges RE vagy RF utasításokat beadhatja. A lista elcsúsztatott REPEAT hurkai mindezt könnyen ellenőrizhetővé teszik. Majdnem minden esetben eggyel több RF utasítás szükséges. Használja az U-t és Z-t szükség szerint. Az END beadása után az "ACCEPTED" (elfogadva) üzenet jelentkezik, majd egymás után a (P)ROC? és (C)OPY? kérdések, jelezvén, hogy a program eljárássá konvertálható a P beadásával (ezt az eljárásnév és szám követi) és/vagy kinyomtatható a C megnyomásával.

3. A program kilistázása (L)

Az L utasításra az aktuális program kilistázódik, és ENTER után a szokásos opciók szerint a program eljárássá alakítható vagy kinyomtatható.

4. A program szerkesztése (E)

Az operációs rendszernek ez a lehetősége megengedi a program megváltoztatását az utasítások törlésével vagy beszúrásával. Kezdje a sorszám beadásával!

A törlések vagy helyettesítések az előírt sorban történnek. A beszúrások az előírt sor elé kerülnek úgy, hogy használjon eggyel nagyobb sorszámot, mint az utolsó soré, ha további utasításokat akar a program végéhez fűzni. Érvénytelen sorszám esetén visszatér a menü [(€)OPY?-n keresztül], ha a sorszám kisebb volt, mint 999. (Téves beadásokat a Snail Logo nem hajt végre.)

A sorszám megadása után a Replace, Insert vagy Delete opció választható. Adjon be R-t, I-t vagy D-t! R és I esetén az "Inserted instructions" felirat tűnik fel, és az utasítások a szokott módon beadhatók. Figyelje meg, hogy a beadott utasítások I-től kezdve lesznek sorszámozva!

Az "Inserted instructions" utasítások sorozatát END-del fejezze be, hogy visszatérjen az Edit (E) sorszámot választó szintjére! Figyelje meg, hogy az END-re akkor is szükség van, ha nem történt utasításbeszúrás. Ezek után másik sorszám választható. Mikor az összes változtatás kész, Z-vel lehet a menüre visszatérni. A menü a (C)OPY? kérdésen keresztül tér vissza.

Figyelje meg, hogy tetszőleges számú utasítás beszúrható vagy törölhető, valamint egyedi RE és RF utasítások is beadhatók vagy törölhetők szerkesztés (E) esetén! (Programbeadáskor viszont csak RE, RF párok vannak elfogadva.) Figyelmeztetés történik ugyan, de ily módon hibás RE, RF sorozatok hozhatók létre a programszerkesztésnél. Természetesen a programok futtatása ilyenkor hibaüzenethez vezet.

Egy, a PROCEDURE utasításban megnevezett eljárás törölhető a programból, ha a számát másik eljárásdefiníció használja. Ha a szerkesztett programban a PR utasítás egy már nem létező eljárás nevével szerepel, akkor hibaüzenet jelenik meg, és a PR utasítás törlődik a program szerkesztett változatából.

5. Eljárásbeadás (P)

Ezzel a lehetőséggel az eljárások közvetlenül beírhatók. Először az eljárás nevét kell megadni, majd a számát 1-től 9-ig. Bármely másik szám esetén a névbeadástól kell ismételni. Ha a név vagy a szám beadása elmarad, akkor a menü jelentkezik be. Ha több, mint egy eljárást használunk, akkor ezeknek különböző számokat kell adni, mivel az azonos sorszámú eljárások közül az utolsó felülírja az előbbit.

Ha két eljárásnak a neve azonos, akkor a kisebb sorszámú kerül sorra, amennyiben a programból hivatkozás történik a névre. Az eljárásokat definiálni, azaz elnevezni, megszámozni és megadni kell, mielőtt egy PR utasításban hivatkoznánk rájuk. (Ez lehet egy egyszerű END utasítás egy „üres" eljárásnál.)

Ezután az eljárás megváltoztatható azonos néven és számon beadott új utasításokkal. Ha a rendszer nem ismer egy eljárást, hibaüzenetet ad a név használatakor.

Figyelje meg, hogy egy eljárás új változatának beadásánál ugyanazt a számot kell használni, mert ez a szám már rögzítve van az eljárás meghívásánál. Még ha az eljárás törölve is van (l. később), a hívó PROCEDURE utasítás azonos számot fog használni és a nem létező eljárást „üres" eljárásnak veszi.

Az összes utasítás, így az U és Z parancsok is, a programbeadásnál látott módon használhatók. Ugyanúgy, mint a programoknál, az RE és RF utasításoknak azonos számban kell szerepelniük az eljárásokon belül. Az eljárásokat szintén, akár a programokat, END-del kell befejezni, és ezután nyomtathatók.

6. Eljárás törlése (C)

Ez a parancs törli az eljárást a nevének megadása után. Ha több azonos nevű eljárás létezik, akkor a legkisebb sorszámú törlődik csupán. Nem tanácsos sok, nem használt eljárást a rendszerben hagyni, mivel ezek terhelést jelentenek és bizonyos műveletek sebességét lecsökkentik.

7. Eljárás kilistázása (K)

Ezzel a megnevezett eljárást a nyomtatón ki lehet listázni.

8. Eljárásnevek listázása (N)

Lehetőséget ad a névvel és számmal adott eljárás kilistázása.

9. Program futtatása (R)

Ez a parancs végrehajtja a programot. Hibaüzenet jelenik meg, ha lehetetlen utasítás vagy nem azonos számú REPEAT és RFINISH utasítás található, vagy nullával való osztás történt, ill. ha betelik a rekurziós verem, vagy egy REPEAT hurok IFEND nélkül áll. A programfutás az aktuális utasítás végrehajtása után a Z megnyomásával állítható meg. A programfutás végén az ábra a nyomtatóval másolható a (C)OPY? kérdésre beadott C válasszal.

10. Visszaverődés/áthaladás (B)

Lehetővé teszi, hogy a rajzterület falai visszaverjék a Csiga nyomát a megfelelő szögben. Ez a Snail Logonak nem éppen túl komoly használata, de jó mulatság lehet. Ha a nyom eléri a falat és a (B)ounce nincs beállítva, akkor a program végrehajtása a WALL üzenettel megszakad. Amikor a visszaverődés be van állítva, akkor a keret színe sárga, egyébként kék.

11. Utasítások (I)

Ez a parancs egymást követő lapokon felsorolja a Snail Logo utasításokat. A lapozás az ENTER billentyű ismételt lenyomásával érhető el, végül pedig a menüre tér vissza. M beadására a menü azonnal visszakapható.

3.3. Összefoglalás

3.3.1. A Snail Logo utasítások összefoglalása

A következő utasításdefiníciókban n-nek, V-nek és N-nek a következő jelentése van:

n valós szám,
V változónév A-tól H-ig
N egész szám.

A számok maximális nagysága utasítástól függően különböző, és a következőkben van definiálva.

FORWARD
Előírt számú lépést tesz az aktuális irányba.

BACKWARD N
Előírt számú lépést tesz az aktuális iránnyal ellentétesen.

RIGHT N
Az aktuális irányt az óramutató járásával megegyezően, az előírt szöggel elforgatja.

LEFT N
A RIGHT-hoz hasonló, csak az óramutató járásával ellentétesen.

REPEAT N
Az ezt követő utasításokat az előírt számban megismétli.

RFINISH
Lezárja a REPEAT hatását.

POSITION n, n
A Csiga aktuális helyzetét az adott értékekre (vízszintes, függőleges) állítja.

COLOUR N
Beállítja a Csiga nyomának színét.

SET V, n
Beállítja a változó értékét.

DECREASE v, n
n-nel csökkenti a változó értékét.

INSREASE V, n
n-nel megnöveli a változó értékét.

SHOW V
A képernyőre kiírja a változó értékét és nevét.

MAKE V=V+V
Változókat ad össze.

MAKE V=V–V
Változókat von ki.

MAKE V=V*V
Változókat szoroz (A szorzójel "*", mint a Basic-ben).

MAKE V=V/V
Változókat oszt el.

IFEND V=V
Programot vagy eljárást fejez be, ha a feltétel teljesül.

IFEND V>V
Programot vagy eljárást fejez be, ha a feltétel teljesül.

IFEND V<V
Programot vagy eljárást fejez be, ha a feltétel teljesül.

SNAIL
Az utána következő utasítások hatására létrejött nyom végére a Csiga-biga szimbólumát rajzolja fel.

NSNAIL
Törli a SNAIL hatását.

DOWN
A Csiga nyoma és szimbóluma ezt követően látható lesz.

UP
A Csiga nyoma és szimbóluma ezt követően láthatatlan.

OUTPUT
A Csiga pályájának egy elemét az adott helyzetben kinyomtatja.

CENTRE
Az aktuális helyzetet a képernyő közepére mozgatja.

NORTH
Az aktuális irányt a kpéernyő felső része felé állítja.

RNORTH
Az aktuális irányt a program vagy eljárás kezdetén felvett értékre állítja vissza.

CLEAR
Törli a képernyőt.

PROCEDURE
Név

PROCEDURE
Név, N, N, N (1, 2 vagy 3 n-et lehet előírni.)

A PROCEDURE utasítás aktivizálja a névvel már előzőleg definiált utasítássorozatot. Szükség esetén egy, két vagy három paraméterérték átadható

az eljárásnak, a név után vesszőkkel elválasztva, ahogy látható. Az eljáráson belül a paraméterek A-val, B-vel és C-vel vannak azonosítva. A paraméterértékek a PROCEDURE utasításban mindig A, B, C sorrendben vannak kijelölve.

END – programok és eljárások szükségszerű utolsó utasítása.

Utasítások	*n és N értéktartománya*
FORWARD, BACKWARD REPEAT, SET INCREASE, DECREASE	Egész számok 0-tól 999-ig,
RIGHT, LEFT PROCEDURE	Egész számok 0-tól 999-ig,
POSITION	1-től 62-ig a vízszintes, 1-től 42-ig a függőleges pozícióra.
COLOUR	Egész számok 0-tól 6-ig.

3.3.2. Megjegyzések az utasítások használatához

1. Minden utasítás lerövidíthető az első két betűjére, kivétel az END.

2. A REPEAT utasításokat PFINISH utasítások kell, hogy kövessék. A REPEAT hurkok egymásba ágyazhatók, és a többszörös hurkok mind a programban, mind az eljárásokban használhatók.

3. A változók legelső értéke zérus.

4. A Csiga szimbólum eltávolítható azonos helyen való újraírással. Egy adott helyen pedig az SN FO 0 utasítássorozattal ábrázolható. (Próbálja ki az SN FO 0 FO 0 utasítások hatását!).

5. Amikor az eljárások az A, B és C változókat használják, ezek értékét a PR utasításban az eljárás hívásánál be kell állítani. A-nak és B-nek akkor is értéket kell adni, ha csupán a C-t kell konkrétan megadni.

Hasonlóképpen az A-t is meg kell adni, ha az eljárás csupán a B-t használja, hogy a rögzített A, B, C sorrend megmaradjon.

6. Mielőtt egyetlen utasítás meg lenne adva a programban, a következő feltételek érvényesek: CENTRE, NORTH, DOWN, NSNAIL, COLOUR 0.

7. A színek számának megadása a számítógép kezelési utasítása szerint történik.

8. RE 0 és RE 1 egyszer hajtják végre az utasításokat.

3.3.3. Működési korlátok

1. A példák és néhány saját program futtatása megmutatja, hogy milyen rövid egy tipikus Snail Logo program. Ennek fényében kell tekinteni a következőket:

Képernyőméret: 62 lépésX42 lépés.
Maximális eljáráshossz: 15 utasítás vagy 70 karakter.
Maximális programhossz: 45 utasítás vagy 200 karakter. Eljárások maximális száma: 9.
Eljárásnevek maximális hossza: 7 karakter (csak betűk).
A karakterek maximális száma a paraméterértékekben 12, beleértve a tizedespontot is.
A maximális számú eljáráshívás egy programban vagy eljárásban 10.
Az egymásba ágyazott ciklusok maximális száma 15 (programban vagy eljárásban). Ezek a maximumok ritkán jelentenek korlátozást.

Az utasítások rövidített formája a normális, és így a 70/200-as határ bőségesen elég. Egy átlagos program 20 utasításnál rövidebb, így elfér egy képernyőre. Egy lapozási módszer a kivételes esetekre rendelkezésre áll.

2. Kivételesen hosszú utasítások, amikor egymásba ágyazott ciklusokban vannak (ezek elcsúsztatva kerülnek kinyomtatásra), túlcsordulhatnak a képernyő következő sorába. Ez nem okoz problémát sem a program listázásakor, sem szerkesztéskor, hacsak több nincs belőlük, miközben a képernyő tele van, és a 200 karakteres határt még a program nem érte el. Ilyenkor a képernyőn túlcsordulás következhet be, és a Basic "Continue" utasítására van szükség, bár mindez ritkán fordul elő.

3.3.4. Hibaüzenetek

Bizonyos esetekben hiba- és figyelmeztető utasítások jelennek meg. Minden esetben az ENTER billentyűvel törölje az üzenetet! A teendők az üzenettől függenek, és a következőkben vannak megadva.

1. Utasítások szintaxishibái

COMMAND ERR
Az utasítás parancsszava nem azonosítható.

NUM IN COMMD
Számjegy található az utasításparancsban. Általában a szóköz elhagyása okozza.

NO SPACE REQ
Felesleges szóköz van az utasításban.

NUM/NAME ERR
Nem megfelelő karakter található az utasítás azon részében, amely számjegy, paraméterérték vagy eljárásnév.

NUM NO GOOD.
A 3.1. szakaszban megadott tartományon kívüli számérték fordul elő.

NUM/NAME REQ.
Egy szám vagy az A, B, C paraméter, vagy egy eljárásnév nincs megadva.

NUMBER??
Számtól különböző karakter található egy számjegy helyén.

NOT VAR NAME.
A, B, ... H-tól különböző betű van a változónév helyén.

PROCEDURE??
Az eljárás neve nem felismerhető.
Az összes utasításhiba esetén a kérdéses utasítás törlődik a hibaüzenettel együtt az ENTER használatával.

2. Egyéb hibaüzenetek és figyelmeztetések

10 PROCS MAX
11 PROCEDURE utasítás használata volt megkísérelve.

REP LOOP ERROR
Érvénytelen REPEAT/PFINISH sorozat volt található programfutás közben, azaz pl. RF ... RE vagy az RE és RF különböző számúak.
A futás félbeszakad és az ENTER a menüre tér vissza.

REPEAT LOOP ONLY PART EDITED
A szerkesztő (Editor) figyelmeztető üzenete az előbbi probléma megelőzésére.

ONLY 15 INSTRS IN PROCEDURES
15 END-től különböző utasítás volt egy eljáráson belül, vagy egy programban az eljárásba konvertáláskor. (Ezért nem lenne hely az END utasítás részére.)

NO SPACE LEFT
Akkor fordul elő, ha egy eljárás túllépi a 70 vagy egy program meghaladja a 200 karaktert. Rövidített utasításnevek használatánál nem valószínű. A program vagy eljárás ENTER-re törlődik.

PROC; NAME CANCELLED
Az előző két hibaüzenet törlésére beadott ENTER után jelentkezik.

PAGE FULL/CONT
Akkor lép föl, ha a program túllépi a 20 utasítást. Az ENTER használatára új oldal nyílik, ez a CONT-tal kezdődik. A sorszámozás azonban 1-től kezdődik, úgyhogy a szerkesztő egymást követő oldalakon nem használható. Maximum 45 utasításból álló program (3 oldalon) adható be és futtatható.

WALL
Akkor jelentkezik, ha a program futása közben a Csiga a rajzterület szélébe ütközik. (Még akkor is, ha a Csiga nem látható ezalatt!) Az ENTER menüre tér vissza.

RF NOT EQUAL RE
L. a 2.3. szakasz 2. pontját.

END OF RECURSION STACK
Az eljárások önmagukat legfeljebb csak 50-szer hívhatják.

REP LOOP IFEND
Egy Repeat ciklus IFEND utasítással volt lezárva. A program végrehajtása nem folytatható.

DIVIDE BY ZERO
Nullával vagy nulla értékű változóval való osztás volt kijelölve.

4. Az egyes fejezetek szimbólumai és a hozzá tartozó programok

```
TO TEKNOC
 RIGHT 45 FAROK
 RIGHT 45 HAT
 LABAK
 RIGHT 90 NYAK
 FEJ
END

TO FAROK
 FORWARD 20
END

TO HAT
 FORWARD 74
 LEFT 90
 GORBE 2 ( - 3 ) 180
 LEFT 90
END

TO LABAK
 PENUP BACK 26 RIGHT 90 FORWARD 6 LEFT 90
 REPEAT 2 [FORWARD 40 PENDOWN Z PENUP]
 LEFT 90 FORWARD 6 RIGHT 90 FORWARD 26 PENDOWN
END

TO Z
 FORWARD 16
 RIGHT 140 FORWARD 24
 LEFT 140 FORWARD 16
 PENUP BACK 16 RIGHT 90 BACK 16 LEFT 90 PENDOWN
END

TO NYAK
 GORBE 2 ( - 8 ) 180
END

TO FEJ
 GORBE 2 8 360
 PENUP RIGHT 90 FORWARD 10 LEFT 90 PENDOWN
 SZEM
END
```

```
TO SZEM
 GORBE 2 15 360
END

TO GORBE :LEPES :FORDULAT :IV
 REPEAT :IV / ABS :FORDULAT [FORWARD :LEPES RIGHT :FORDULAT]
END

TO ABS :X
 IF 0 > :X THEN OUTPUT :X * ( - 1 ) ELSE OUTPUT :X
 ND


TO ALMA
 LEFT 110
 FEL.ALMA
 NYEL
 FEL.ALMA
 CSUTAK
END

TO FEL.ALMA
 GORBE 2 3 260
END

TO NYEL
 LEFT 170 FORWARD 10
 GORBE 2 5 110
 RIGHT 90 FORWARD 10 RIGHT 80
 GORBE 2 ( - 6 ) 110
 LEFT 120
END

TO CSUTAK
 LEFT 140 Z
 RIGHT 170 FORWARD 8
 LEFT 135 Z
END

TO Z
 FORWARD 8 RIGHT 140
 FORWARD 12 LEFT 140
 FORWARD 8
END

TO ABS :X
 IF 0 > :X THEN OUTPUT :X * ( - 1 ) ELSE OUTPUT :X
END
```

```
TO COM
 LEFT 130
 CC
 PENUP LEFT 135 FORWARD 8 RIGHT 90 PENDOWN
 VONASOK
END

TO CC
 GORBE 1 3 270
 RIGHT 90 FORWARD 5 RIGHT 90
 GORBE 1 ( - 4 ) 275
 RIGHT 90 FORWARD 5
END

TO VONASOK
 X 90
 PENUP RIGHT 135 FORWARD 10 RIGHT 90 PENDOWN
 X - 90
END

TO VON
 PENUP LEFT 135 FORWARD 8 RIGHT 90 PENDOWN
 X 90
 PENUP RIGHT 135 FORWARD 10 RIGHT 90 PENDOWN
 X - 90
END

TO X :X
 FORWARD 12
 LEFT 1.5 * :X FORWARD 5
 LEFT :X / 2 FORWARD 5
 LEFT :X / 2 FORWARD 5
END


TO SNAIL
 HAZ
 TORZS
 CSAP
END

TO HAZ
 SPIRAL 4 2 20
END

TO SPIRAL :MERET :NOV :MAX
 IF :MERET > :MAX THEN STOP
 RIGHT 61 FORWARD :MERET
 SPIRAL :MERET + :NOV :NOV :MAX
END

TO TORZS
 SETHEADING 270
 FORWARD 50
END

TO CSAP
 RIGHT 60 FORWARD 20
 BACK 20
 RIGHT 60 FORWARD 20
END
```

```
TO SPEKTRUM
 SPEKT 3
 SPEKT 2
 SPEKT 1
END
```

```
TO SPEKT :M
  SETHEADING 30
  REPEAT 5 [ FORWARD 60 LEFT 120 FORWARD 3 / :M
  LEFT 60 FORWARD 20 RIGHT 60 FORWARD 3 / :M
   RIGHT 120 ]
END
```

5. Fordítóprogram

A könyv Appleprogramjait Commodore-on is kipróbálhatja az Olvasó. Erre két lehetőség kínálkozik.

1) Az Apple programban levő utasítások C-64 Terrapin megfelelőit kikeresi a 6. Függelék táblázatából.

2) Használja a következő programot. Nyilvánvaló, hogy ezt a lehetőséget azoknak kínáljuk, akik könyvünk második részét C-64 géppel kívánják végigkövetni. A programok beírása után a fordítót mentse háttértárra, így bármikor ismét használhatja.

```
TO #FORDITAS :LISTA
 IF EMPTY? :LISTA THEN STOP
 #FORDITO ( LIST FIRST :LISTA )
 #FORDITAS BF :LISTA
END

TO #FORDITO :L
 MAKE "LOG "FALSE
 MAKE "LOGO "FALSE
 MAKE "LARC "FALSE
 MAKE "L.PR "FALSE
 MAKE "L.TH "FALSE
 MAKE "L.CR "FALSE
 MAKE "L.LR "FALSE
 IF :L = [] THEN STOP
 DEFINE FIRST :L #FRD TEXT FIRST :L
END

TO #FRD :PL
 IF :PL = [] THEN OP []
 OP SE #F #FRD BF :PL
END
```

```
TO #F
 TEST LIST? FIRST :PL
 IFT IF :LOG THEN MAKE "LOG "FALSE OP #FR!
D FIRST :PL ELSE OP #MICS
 IFF TEST :L.CA
 IFT MAKE "L.CA "FALSE OP [RUN ]
 IFF TEST :L.TH
 IFT MAKE "L.TH "FALSE
 IFT IF FIRST :PL = ""TOPLEVEL THEN OP [T!
OPLEVEL ] ELSE OP "
 IFF TEST :L.LA
 IFT MAKE "L.LA "FALSE OP WORD BF FIRST :!
PL ":
 IFF OP #CSERE FIRST :PL
END

TO #MICS
 LOCAL "SX
 TEST NOT EMPTY? FIRST :PL
 IFT TEST "IF = FIRST FIRST :PL
 IFT MAKE "SX #IFFELD FIRST :PL
 IFT IF :LOGO THEN MAKE "LOGO "FALSE OP F!
PUT BL :SX LAST :SX ELSE OP ( LIST :SX )!

 IFF OP ( LIST #FRD FIRST :PL )
END

TO #CSERE :W
 IF MEMBER? :W [IFF IFFALSE IFT IFTRUE SE!
TCURSOR SETPOS TOWARDS ] THEN MAKE "LOG !
"TRUE
 IF :W = "ARCTAN THEN MAKE "LARC "TRUE
 IF :W = ".PRINTER THEN MAKE "L.PR "TRUE
 IF :W = "THROW THEN MAKE "L.TH "TRUE
 IF :W = "LABEL THEN MAKE "L.LA "TRUE
 IF :W = "CATCH THEN MAKE "L.CA "TRUE
 TEST THING? ( WORD "# :W )
 IFT OP THING ( WORD "# :W )
 IFF TEST :LARC
 IFT MAKE "LARC "FALSE OP SE :W 1
 IFF TEST :L.PR
 IFT MAKE "L.PR "FALSE OP "
 IFF TEST ( ANYOF :L.TH :L.LA :L.CA )
 IFT OP "
 IFF OP :W
END

 TO #IFFELD :L
  IF :L = [] THEN OP []
  TEST NOT LIST? FIRST :L
  IFT OP SE ( #CSERE FIRST :L ) ( #IFFELD !
 BF :L )
  IFF TEST 1 = COUNT :L
  IFT OP SE "THEN FIRST #FRD :L
  IFF TEST LIST? ITEM 2 :L
  IFT OP LPUT #IFO BF BF :L ( SE "THEN FIR!
 ST #FRD ( LIST FIRST :L ) "ELSE FIRST #F!
 RD ( LIST ITEM 2 :L ) )
  IFF OP LPUT #IFO BF :L SE "THEN FIRST #F!
 RD :L
 END
```

```
TO #IFU :U
 IF EMPTY? :U THEN OP " ELSE MAKE "LOGU "!
TRUE OP ( LIST #FRU :U )
END
```

A Fordítói szótárt a következő utasítások begépelésével lehet létrehozni.

```
MAKE "#AND "ALLOF
MAKE "#ARCTAN "ATAN
MAKE "#BUTTONP "PADDLEBUTTON
MAKE "#CLEAN "CS
MAKE "#CLEARSCREEN "DRAW
MAKE "#CURSOR "CURSORPOS

MAKE "#EDNS [ED NAMES]
MAKE "#EMPTYP "EMPTY?
MAKE "#ERALL [ER ALL]
MAKE "#ERN [ERNAME]
MAKE "#ERNS [ER NAMES]
MAKE "#ERPS [ER PROCEDURES]

MAKE "#FENCE "WRAP
MAKE "#INT "INTEGER

MAKE "#KEYP "RC?
MAKE "#LISTP "LIST?
MAKE "#LOAD "READ
MAKE "#MEMBERP "MEMBER?
MAKE "#NAMEP "THING?
MAKE "#NODES ".NODES

MAKE "#NUMBERP "NUMBER?
MAKE "#OR "ANYOF
MAKE "#POALL [PO ALL]
MAKE "#PONS [PO NAMES]
MAKE "#POPS [PO PROCEDURES]
MAKE "#READCHAR "RC
MAKE "#READLIST "REQUEST
MAKE "#RECYCLE ".GCOLL
MAKE "#RERANDOM "RANDOMIZE
MAKE "#SETBG "BG
MAKE "#SETCURSOR "CURSOR
MAKE "#SETPC "PC
MAKE "#SETPOS "SETXY
MAKE "#SETSCRUNCH ".ASPECT
MAKE "#TEXTSCREEN "NODRAW
MAKE "#TYPE "PRINT1
MAKE "#WORDP "WORD?
MAKE "#.PRINTER "PRINTER


MAKE "#SHOW "FPRINT
```

Bizonyos Apple Logo utasítások még mindig hiányoznak, pontosan azok, amelyeknek egyáltalán nincs a Terrapin Logoban megfelelője. Ezek a következők:

BACKGROUND
BURY
CATCH
COPYDEF
DEFINEDP
DISK
DOT
ERROR
EQUALP
GTPROP
NAME
PACKAGE
PEN
PENCOLOR
PENREVERS E
POS
PKGALL
PLIST
PPROP
PPS
PRIMITIVEP
PRODUCT
PEMPROP
REPARSE
SCRUNCH
SETPEN
SHOWNP
SUM
THROW
UNBURY
WINDOW
.BPT

Néhányat ezek közül könnyű előállítani egy megfelelő eljárást írva:

```
TO PRODUCT :X :Y
 OP :X * :Y
END

TO EQUALP :X :Y
 OP :X = :Y
END

TO SUM :X :Y
 OP :X + :Y
END

TO ERROR
 OP []
END

TO DEFINEDP :X
 IF EMPTY? TEXT :X THEN OP "FALSE ELSE OP "TRUE
END

TO NAME :X :Y
 MAKE :Y :X
END
```

Példaként a BUBBLESORT rendezést fordítottuk át.

```
TO BUBBLE.SORT :NEV
 LOCAL "I MAKE "I 0
 LOCAL "VEGE MAKE "VEGE LAST DIMV :NEV
 REPEAT :VEGE [MAKE "I :I + 1 LEGKISEBB.ELORE :VEGE :I]
END

TO LEGKISEBB.ELORE :VEGE :KEZD
 LABEL "START
 IF ( OV :NEV 1 :VEGE - 1 ) > ( OV :NEV 1!
 :VEGE ) [CSERE ] MAKE "VEGE :VEGE - 1
 IF :VEGE > :KEZD [GO "START ]
END

TO CSERE
 LOCAL "H MAKE "H OV :NEV 1 :VEGE
 IV :NEV 1 :VEGE OV :NEV 1 ( :VEGE - 1 )
 IV :NEV 1 :VEGE - 1 :H
END


TO DEFV :NEV :Z :S
 FOR "I 1 :Z 1 [MAKE (WORD "V :NEV :I ) 0!
SZLOP :S ]
 MAKE ( WORD "V :NEV ) SE :Z :S
END
```

```
TO USZLOP :S
 LOCAL "SP MAKE "SP []
 REPEAT :S [MAKE "SP SE :SP ".]
 OP :SP
END

TO DIMV :NEV
 OP THING WORD "V :NEV
END

TO OV :NEV :Z :S
 CATCH "ERROR [OP ITEM :S THING (WORD "V :NEV" :Z)]
 HIBA
END

TO HIBA
 LOCAL "HIBA MAKE "HIBA ERROR
 IF EMPTYP :HIBA [STOP]
 PR [HIBAS INDEX ?]
 PR :HIBA
 THROW "TOPLEVEL
END

TO SZ1.MOGE.SZ2 :SZ1 :SZ2
 IF EMPTYP :SZ1 [OP "TRUE ]
 IF EMPTYP :SZ2 [OP "FALSE ]
 IF FIRST :SZ1 = FIRST :SZ2 [OP SZ1.MOGE.!
SZ2 BF :SZ1 BF :SZ2 ] OP ASCII FIRST :SZ!
1 < ASCII FIRST :SZ2
END

TO IV :NEV :Z :S :SZO
 CATCH "ERROR [MAKE (WORD "V :NEV :Z ) BE!
ILLESZ! THING (WORD "V :NEV :Z) :S :SZO !
]
 HIBA
END

TO BEILLESZ! :V :S :SZO
 IF :S = 1 [OP SE :SZO BF :V]
 OP SE FIRST :V BEILLESZ! BF :V :S - 1 :SZO
END

TO NYUMV :NEV
 FOR "I 1 FIRST! DIMV :NEV 1 [FOR "J 1 LAS!
T DIMV :NEV 1 [TYPE OV :NEV :I :J TYPE "!
 ] PR " ]
END

TO FOR :VAR@ :A@ :E@ :S@ :UTASITAS
 LOCAL :VAR@
 MAKE :VAR@ :A@
 REPEAT SUM 1 ( :E@ - :A@ ) / :S@ [RUN :U!
TASITAS MAKE :VAR@ :S@ + THING :VAR@ ]
END
```

Begépelve ezt a listát, a fordítás a következőképp érhető el:

```
#FORDITAS    [BUBBLE.SORT
             LEGKISEBB.ELORE
             CSERE
             SZ1.MOGE.SZ2
             DEFV
             OSZLOP
             DIMV
             OV
             HIBA
             IV
             BEILLESZT
             NYOMV
             FOR]
```

Egy-két perces várakozás után a fordítás elkészül. Ekkor a munkamezőben a C-64-nek megfelelő eljárásokat találjuk.

```
TO BUBBLE.SORT :NEV
 LOCAL "I MAKE "I 0
 LOCAL "VEGE MAKE "VEGE LHST DIMV :NEV
 REPEHT :VEGE [MAKE "I :I + 1 LEGKISEBB.E!
LORE :VEGE :I ]
END



TO LEGKISEBB.ELORE :VEGE :KEZD
 START:
 IF ( OV :NEV 1 :VEGE - 1 ) > ( OV :NEV 1!
 :VEGE ) THEN CSERE
 MAKE "VEGE :VEGE - 1
 IF :VEGE > :KEZD THEN GO "START
END


 TO CSERE
  LOCAL "H MAKE "H OV :NEV 1 :VEGE
  IV :NEV 1 :VEGE OV :NEV 1 ( :VEGE - 1 )
  IV :NEV 1 :VEGE - 1 :H
 END

TO SZ1.MOGE.SZ2 :SZ1 :SZ2
 IF EMPTYP :SZ1 THEN OP "TRUE
 IF EMPTYP :SZ2 THEN OP "FALSE
 IF FIRST :SZ1 = FIRST :SZ2 THEN OP SZ1.M!
OGE.SZ2 BF :SZ1 BF :SZ2
 OP ASCII FIRST :SZ1 < ASCII FIRST :SZ2
END
```

```
TO DEFV :NEV :Z :S
 FOR "I 1 :Z 1 [MAKE (WORD "V :NEV :I ) OSZLOP :S]
 MAKE ( WORD "V :NEV ) SE :Z :S
END

TO OSZLOP :S
 LOCAL "SP MAKE "SP []
 REPEAT :S [MAKE "SP SE :SP ",]
 OP :SP
END

TO DIMV :NEV
 OP THING WORD "V :NEV
END

TO OV :NEV :Z :S
 RUN [OP ITEM :S THING (WORD "V :NEV :Z)]
 HIBA
END

TO HIBA
 LOCAL "HIBA MAKE "HIBA ERROR
 IF EMPTY? :HIBA THEN STOP
 PR [HIBAS INDEX ?]
 PR :HIBA
 TOPLEVEL
END

TO IV :NEV :Z :S :SZO
 RUN [MAKE (WORD "V :NEV :Z ) BEILLESZT T!
HING (WORD "V :NEV :Z) :S :SZO ]
 HIBA
END

TO BEILLESZT :V :S :SZO
 IF :S = 1 THEN OP SE :SZO BF :V
 OP SE FIRST :V BEILLESZT BF :V :S - 1 :SZO
END

TO NYOMV :NEV
 FOR "I 1 FIRST DIMV :NEV 1 [FOR "J 1 LAS!
T DIMV :NEV 1 [PRINT1 OV :NEV :I :J PRIN!
T1 " ] PR " ]
END

TO FOR :VAR@ :A@ :E@ :S@ :UTASITAS
 LOCAL :VAR@
 MAKE :VAR@ :A@
 REPEAT SUM 1 ( :E@ - :A@ ) / :S@ [RUN :U!
TASITAS MAKE :VAR@ :S@ + THING :VAR@ ]
END
```

Felhívjuk a figyelmet, hogy még egy apró módosításra szükség lesz, ugyanis a TYPE'' hatására szóköz kerül a számok közé, míg a PRINT1'' nem tesz ki szóközt.

6. Személyi számítógépek Logo változatainak táblázata

A következőkben felsoroljuk a LCSI Logo funkciókat, és – amennyiben eltér is a nevük – akkor mellé írjuk a Spectrum Logo, Snail Logo és Terrapin Commodore Logo változatok megfelelő szavait. Azonosság esetén X jelet írunk, ha a funkció nem létezik, akkor üresen marad a helye. A változatok editáló funkciói kissé eltérőek, a vezérlőgombok szerepe más. Ezzel kapcsolatban az adott változat kézikönyvét kell tanulmányozni. A Spectrum Logo, mint a táblázatból is kitűnik, csak minimálisan tér el az LCSI Apple Logotól. Így külön melléklet nem készült róla. Ezen áttekintés segítséget kíván nyújtani ahhoz, hogy a könyvben szereplő LCSI Logo példákat könnyen átírhassuk az említett Logo változatokra, hivatkozva arra az oldalszámra, ahol a megfelelő utasítás alakját megtalálhatjuk.

LCSI Apple Logo	Spectrum Logo	Snail Logo	Terrapin C-64
*	X	X	X
+	X	X	X
–	X	X	X
.BPT			
.CONTENTS			X
.DEPOSIT	X		X
.EXAMINE	X		X
.PRINTER	PRINTON		PRINTER
/	X	X	X
;	X		X
<	X	X	X
=	X	X	X
>	X	X	X
AND	X		ALLOF
ARCTAN	X		ATAN
ASCII	X		X

LCSI Apple Logo	Spectrum Logo	Snail Logo	Terrapin C-64
BACK (BK)	X	X	X
BACKGROUND (BG)	X		
BURY			
BUTFIRST (BF)	X		X
BUTLAST (BL)	X		X
BUTTONP			PADDLEBUTTON
CATALOG	X		X
CATCH			
CHAR	X		X
CLEAN	X		CLEARSCREEN,CS
CLEARSCREEN (CS)	X		DRAW
CLEARTEXT	X, CT		NODRAW
CO			X, CONTINUE
COPYDEF	X		
COS	X, COSINE		X
COUNT	X		X
CURSOR			
DEFINE	X		X
DEFINEDP	X		
DISK	-		
DOT	X	OUTPUT	
EDIT (ED)	X		X
EDNS	X		ED NAMES
EMPTYP	X		EMPTY?
EQUALP	X		
ERALL	X		ER ALL
ERASE (ER)	X		X
ERASEFILE	X		X
ERN	X		ERNAME
ERNS	X		ER NAMES
ERPS	X		ER PROCEDURES
ERROR			
FENCE	X		NOWRAP
FIRST	X		X
FORWARD (FD)	X	X	X
FPUT	X		X
FULLSCREEN			X
GO			X
GPROP			
HEADING	X		X
HIDETURTLE (H)	X	NSNAIL	X
HOME	X	CENTRE	X
IF	X		X

LCSI Apple Logo	Spectrum Logo	Snail Logo	Terrapin C-64
IFFALSE (IFF)			X
IFTRUE (IFT)			X
INT	X		INTEGER
ITEM	X		X
KEYP	X		RC?
LABEL			
LAST	X		X
LEFT (LT)	X	X	X
LIST	X		X
LISTP	X		LIST?
LOAD	X		READ
LOCAL			X
LPUT	X		X
MAKE	X	X, SET	X
MEMBERP	X		MEMBER?
NAME			
NAMEP	X		THING?
NODES	X		.NODES
NOT	X		X
NUMBERP	X		NUMBER?
OR	X		ANYOF
OUTPUT (OP)	X		X
PACKAGE			
PADDLE			X
PAUSE			X
PEN			
PENCOLOR	X ,PC		
PENDOWN (PD)	X	DOWN	X
PENERASE	X		X
PENREVERSE (PX)	X		
PENUP	X	UP	X
PKGALL			
PLIST			
PO	X		PRINTOUT, X
POALL	X		PO ALL
PONS	X		PO NAMES
POPS	X		PO PROCEDURES
POS	X, POSITION		
POTS	X		X
PPROP			
PPS			
PRINT (PR)	X		X
PRIMITIVEP	.PRIMITIVES, X		

LCSI Apple Logo	Spectrum Logo	Snail Logo	Terrapin C-64
PRODUCT	X		
QUOTIENT			
RANDOM	X		X
READCHAR	X, RC		READCHARACTER, RC
READLIST	X, RL		REQUEST
RECYCLE	X		.GCOLL
REMAINDER	X		X
REMPROP			
REPARSE			
REPEAT	X	X, RFINISH	X
RERANDOM			RANDOMIZE
RIGHT (RT)	X	X	X
ROUND	X		X
RUN	X		X
SAVE	X		X
SCRUNCH	X		
SENTENCE (SE)	X		X
SET BG	X		BACKEROUND, BG
SETCURSOR	X, SETCUR		CURSOR
SETDISK	SETDRIVE		X
SETHEADING (SETH)	X		X
SETPC	X		PENCOLOR PC
SETPEN			
SETPOS	X	POSITION	SETXY
SETSCRUNCH	X		.ASPECT
SET X	X		X
SETY	X		X
SHOW	X	X	FPRINT
SHOWNP	X		
SHOWTURTLE (ST)	X	SNAIL	X
SIN	X, SINE		X
SPLITSCREEN			X
SQRT	X		X
STOP	X		X
SUM	X		
TEST			X
TEXT	X		X
TEXTSCREEN	X		NODRAW
THROW			
THROW "TOPLEVEL	TOPLEVEL		TOPLEVEL
TO	X		X
TOWARDS	X		X

LCSI Apple Logo	Spectrum Logo	Snail Logo	Terrapin C-64
THING (:)	X		X
TYPE	X		PRINT1
UNBURY			
WINDOW	X		
WORD	X		X
WORDP	X		WORD?
WRAP	X		X
XCOR	X		X
YCOR	X		X
	.BLOAD		BLOAD
	.BSAVE		BSAVE
	.CALL		X
	BYE		GOODBYE
	PRINTOFF		NOPRINTER
	SETTC		TEXTBG
		DECREASE	
		INCREASE	
		IFEND	
		NORTH	
		PROCEDURE	
		RNORTH	
			.CTYO
			.OPTION
			.SPRINT
			BITAND
			BITOR
			BITXOR
			CLEARTEXT
			CLEARINPUT
			DOS
			DRAWSTATE
			ED ALL
			ERASEPICT
			JOYSTICK
			JOYBUTTON
			NOTRACE
			READPICT
			SETSHAPE
			SHAPE
			SINGLECOLOR
			SAVEPICT
			STAMPCHAR
			TEXTCOLOR
			TRACE
			WHO

Irodalom

Abelson, H.: Apple Logo; Mc Graw Hill, 1982.
Abelson – di Sessa: Turtle Geometry; MIT Press 1982.
Paperts, S.: Mindstorms: Children, Computers, and Powerfull Ideas: BASIC Books, 1980.
Ross, P.: Logo Programming, Addison – Wesley, 1983.
Taylor, R. P. ed: The Computer in the School: Tutor, Tool, Turtle; Teacher College Press, 1980.

Cikkek olvashatók e témában:

a BYTE Magazin 1982/VIII. számában
a Mikroszámítógép Magazin 1985/2., 3. számában és a Számítástechnika hasábjain

Kiadja a Műszaki Könyvkiadó
Felelős kiadó: Fischer Herbert igazgató

Révai Nyomda Egri Gyáregység
Felelős vezető: Horváth Józsefné dr.

Műszaki vezető: Kőrizs Károly
Műszaki szerkesztő: Molnár József
A borítót és a kötést tervezte:
A könyv ábráit rajzolta: Orlay Ernő
A könyv formátuma: A/5
Ívterjedelme: 12 (A5)
Ábrák száma: 227
Papír minősége: 65 g. fam. regénnyomó
Betűcsalád és -méret: 10/11° Press Roman
Azonossági szám: 61360
MŰ: 3924–i–8689
A kézirat lezárva 1985. november
Készült az MSZ 5601 és 5602 szerint

Hibajegyzék

A 74. oldal első programja helyesen:

```
TO GORBE2  :MERET  :SZINT
 IF :SZINT =0 THEN FORWARD :MERET STOP
 GORBE2 :MERET / 3 :SZINT - 1
 LEFT 90
 GORBE2 :MERET / 3 :SZINT - 1
 RIGHT 90
 GORBE2 :MERET / 3 :SZINT - 1
 RIGHT 90
 GORBE2 :MERET / 3 :SZINT - 1
 LEFT 90
 GORBE2 :MERET / 3 :SZINT - 1
END
```

A 71. oldal második programja helyesen:

```
TO GORBE2.NEGYZET :MERET
 REPEAT 2 [GORBE2 :MERET 4 RIGHT 180]
END
```

A 303. oldalról hiányzó program:

```
TO SPEKT :M
SETHEADING 30
REPEAT 5[FORWARD 20 LEFT 120 FORWARD 3/:M LEFT 60 FORWARD 20
RIGHT 60 FORWARD 3/:M RIGHT 120]
END
```

83 Ft

Felvilágosítás:
KSH SZÜV
Vállalkozoi és Marketing Osztály
1067 Bp., Lenin krt. 57–59.
Telefon: 223-409
Telex: 227 610